Blind Polynomial Evaluation and Data Trading

Yi Liu[1,3], Qi Wang[1,2], and Siu-Ming Yiu[3]

[1] Guangdong Provincial Key Laboratory of Brain-inspired Intelligent Computation,
Department of Computer Science and Engineering,
Southern University of Science and Technology, Shenzhen 518055, China
liuy7@mail.sustech.edu.cn
wangqi@sustech.edu.cn
[2] National Center for Applied Mathematics (Shenzhen),
Southern University of Science and Technology, Shenzhen 518055, China
[3] Department of Computer Science,
The University of Hong Kong, Pokfulam, Hong Kong SAR, China
smyiu@cs.hku.hk

Abstract. Data trading is an emerging business, in which data sellers provide buyers with, for example, their private datasets and get paid from buyers. In many scenarios, sellers prefer to sell pieces of data, such as statistical results derived from the dataset, rather than the entire dataset. Meanwhile, buyers wish to hide the results they retrieve. Since it is not preferable to rely on a trusted third party (TTP), we are wondering, in the absence of TTP, whether there exists a *practical* mechanism satisfying the following requirements: the seller Sarah receives the payment if and only if she *obliviously* returns the buyer Bob the *correct* evaluation result of a function delegated by Bob on her dataset, and Bob can only derive the result for which he pays. Despite a lot of attention data trading has received, a *desirable* mechanism for this scenario is still missing. This is due to the fact that general solutions are inefficient when the size of datasets is considerable or the evaluated function is complicated, and that existing efficient cryptographic techniques cannot fully capture the features of our scenario or can only address very limited computing tasks.

In this paper, we propose the *first desirable* mechanism that is practical and supports a wide variety of computing tasks — evaluation of arbitrary functions that can be represented as polynomials. We introduce a new cryptographic notion called *blind polynomial evaluation* and instantiate it with an explicit protocol. We further combine this notion with the blockchain paradigm to provide a *practical* framework that can satisfy the requirements mentioned above.

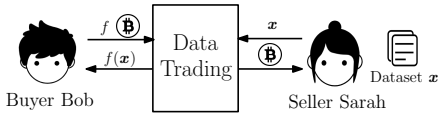
Keywords: Blind polynomial evaluation · Blockchain · ElGamal encryption · Encryption switching protocol · Paillier encryption.

1 Introduction

Nowadays, data trading is an emerging business, which may involve different kinds of data, such as financial data, commercial data, and personal data. As

the public gradually realizes the value of data, data trading has attracted more and more attention. Traditional data trading trivially seeks help from a central platform as a trusted third party (TTP). In most trading strategies, sellers send the entire dataset directly to buyers, which is fairly exchanged under the coordination of the central platform.

Unfortunately, in these solutions, central platforms are heavily relied on and have to be assumed to act honestly. Once the central platform is corrupted, the interests of parties will be significantly hurt. Furthermore, in many data trading scenarios, sellers may prefer to sell only certain calculation results on the dataset rather than exposing the entire dataset *at one time*. At the same time, buyers in a blind fashion wish to hide from sellers the results they retrieve. Therefore, it would be preferable to design a practical mechanism for selling only function evaluation results and capturing requirements like correctness, privacy, consistency, and fairness, but without the existence of TTP.



An illustration of such a scenario is shown above. More precisely, without the help of TTP, the seller Sarah possessing dataset $\boldsymbol{x}$ receives the payment *if and only if* she *obliviously* helps the buyer Bob possessing a function f get the result $f(\boldsymbol{x})$ *correctly*. Meanwhile, Bob can *only* derive the result for which he pays from the data trading. Besides, we should ensure the *consistency* of datasets in two transactions, *i.e.*, for each transaction (with the same buyer or with different buyers), Sarah should use the same dataset $\boldsymbol{x}$. In addition, for the reason that datasets during data trading tend to remain unchanged and consistent, we prefer a pre-processing procedure that amortizes the processing cost of datasets rather than a one-time solution. Here we call a solution *one-time* if every execution of such a solution involves a new entire processing procedure of the dataset.

However, even if data trading is more and more important, desirable solutions meeting the above requirements are still unknown. Although general solutions, such as Yao's garbled circuits [30] using universal circuit [27, 31] and fully homomorphic encryption [14] (see more discussion in Section 2.2 and Section 2.3) can theoretically be used as a component for this scenario, they are infeasible for *practical use* when the sizes of datasets are considerable and the evaluated functions are complicated. Furthermore, existing efficient frameworks cannot *fully* capture the features of our scenario and practical cryptographic tools can only cover *limited* computing tasks. Thus, despite the existence of theoretical solutions, the following question remains open:

> *How to construct a practical mechanism for the requirements of data trading mentioned above, with the capability of supporting a wide variety of computing tasks?*

We answer this question in the present paper.

1.1 Our Results

In this paper, we focus on arbitrary functions that can be represented as polynomials and propose the *first* practical solution that fulfills all aforementioned requirements. We remark that the evaluation of polynomials is powerful and can be utilized in many applications. To further motivate our results, we illustrate some potential applications from three aspects: (i) Polynomial evaluation supports many statistical numerical calculations, including mean, variance, determinant, inner product, Minkowski distance, etc. (ii) Since many functions can be approximated by *Taylor polynomials*, our work supports approximate evaluations of these functions. (iii) Datasets can be specially designed to support many operations through polynomial evaluation. Here we provide a toy example. Suppose that a seller holds a dataset containing the gender and salary of employees at a company. In cells for gender, female employees are specifically represented by zero and male by one. If a buyer intends to calculate the total salary of employees of a particular gender, he requires selecting and summing the salary items of employees of that gender. We denote the gender and salary as b_i and s_i, respectively, for the term of the index i. Then calculating the total salary of female employees is equivalent to evaluating the polynomial $\sum_i (s_i - b_i s_i)$ on the dataset and that of male employees equals $\sum_i b_i s_i$.

To capture the features of the scenario for polynomial evaluations on private datasets, we introduce a new cryptographic notion called *blind polynomial evaluation*. This notion can be viewed as a subset of two-party computation (2PC) problems [29] and is of independent interest. We further combine our blind polynomial evaluation protocol with the blockchain paradigm to provide a practical solution that achieves fairness of exchange for the scenario of data trading.

Here we briefly introduce the underlying insights. We borrow the idea from [6] of using two compatible homomorphic encryption schemes and introducing a switching mechanism between them to support complicated computing tasks. But we note that the scenario we consider is anyhow different from that of [6] (for a comparison, see Section 2.4). Through this mechanism, buyers can evaluate their polynomials on sellers' encrypted datasets via additively and multiplicatively homomorphic properties simultaneously. This idea is simple, but it is a *highly nontrivial* approach for two main reasons: two schemes should be *compatible*, and switching should be guaranteed to perform *correctly*. For our scenario, we need to ensure that two encryption schemes are switched *secretly* and *correctly* when one party holds the complete private keys and intends to behave dishonestly. In addition, we also consider the fact that almost all multiplicatively homomorphic encryption schemes do not support encryption of zero, but it is indeed required in some scenarios. We customize a new multiplicatively homomorphic encryption scheme to resolve this situation. Furthermore, we introduce how to achieve fair exchange of the final result over blockchain.

We summarize the main contributions of this work in the following.

1. Considering the scenario of data trading, we introduce a new cryptographic notion, namely, blind polynomial evaluation. We propose a generic construc-

tion of this notion with communication cost $\mathcal{O}(k)$, where k is the number of terms of polynomials.
2. We propose a *small-constant-round* protocol to instantiate the generic construction to support polynomial evaluation over $\mathbb{Z}_n^*$, where n is a strong RSA modulus having two distinct prime factors of the same length, under standard computational hardness assumptions. Furthermore, we extend our instantiation from polynomial evaluation over $\mathbb{Z}_n^*$ to that over $\mathbb{Z}_n$.
3. We integrate our blind polynomial evaluation protocol with the blockchain paradigm to support fair exchange in the data trading scenario.
4. We analyze our protocol in terms of both round complexity and a *proof-of-concept* implementation to provide evidence that the protocol is practical.

The rest of the present paper is organized as follows. In Section 2, we introduce some related work, with an emphasis on both relevance and difference compared to our work. We then introduce the notion of blind polynomial evaluation together with a generic construction and corresponding definitions in Section 3. The protocol to instantiate the generic construction over $\mathbb{Z}_n^*$ is presented in Section 4, and is further extended from $\mathbb{Z}_n^*$ to $\mathbb{Z}_n$ in Section 5. In Section 6, we show how to combine our blind polynomial evaluation protocol with the blockchain paradigm to achieve fair exchange for data trading. Finally, analysis of the practicality of our protocol is given in Section 7. All proofs of security can be found in the full version of this paper [18].

2 Related Work

In this section, we recall some classical definitions and results that are related to our work.

2.1 Oblivious Polynomial Evaluation

Naor and Pinkas [22] proposed a cryptographic notion named *oblivious polynomial evaluation* (OPE), and then an extended version called *oblivious multivariate polynomial evaluation* [26] was proposed. These notions are for the scenario that a receiver holding a value x (resp. vector $\boldsymbol{x}$) intends to compute $p(x)$ (resp. $p(\boldsymbol{x})$) with the help of a sender possessing a private univariate (resp. multivariate) polynomial p. By the protocol, the receiver gets only $p(x)$ (resp. $p(\boldsymbol{x})$), and the sender can infer no information about x (resp. $\boldsymbol{x}$) from the interaction.

We note that the scenario of data trading we now consider is different from OPE. On the one hand, the receiver of the evaluation result is the polynomial provider in our setting, while the data provider in OPE. This difference leads to very different security definitions. On the other hand, most OPE protocols are designed for one-time use. Such solutions are not preferable and could lead to data inconsistencies among transactions as we have mentioned in Section 1.

2.2 Universal Circuit and Garbled Circuit

For a fixed universal circuit U_n such that $U_n(x, C) = C(x)$ for any circuits having at most n gates, it is easy to see that universal circuits combined with the garbled circuit technique can theoretically be used for the data trading scenario we consider. However, overhead is prohibitive for the following four main reasons.

- For polynomial evaluation, complicated polynomials lead to boolean circuits with considerable sizes. Meanwhile, representations of boolean circuits using universal circuits involve a significant expansion of the circuit size — $\mathcal{O}(n \log n)$ with significant constant terms as well as the low-order terms.
- To ensure data consistency, a costly consistency check of the entire dataset may need to be encoded in the circuit.
- To be secure against malicious parties, expensive techniques (*e.g.*, cut-and-choose approach) should be involved.
- The solution will be a one-time use solution.

In contrast, our solution is efficient and overcomes all of these issues.

2.3 Homomorphic Encryption

Homomorphic encryption is an encryption scheme that allows computations to be performed on encrypted data, such that the decryption of the final result equals the result directly computed from the plaintexts. As a classic example, ElGamal [13] cryptosystem is multiplicatively homomorphic. For additively homomorphic encryption schemes, one of the well-known schemes is Paillier cryptosystem [23], which supports additions of encrypted values and multiplications between encrypted values and constants. Although these cryptosystems (named *partially homomorphic encryption* (PHE)) are practically used in many applications, they support very limited computing tasks (such as only addition or multiplication) and are limited in many other applications. In 2009, Gentry proposed the first *fully homomorphic encryption* (FHE) [14], from which it is possible to perform arbitrary computations. Following Gentry's seminal work, some FHE schemes are proposed afterward. The noted barrier of FHE is that current FHE schemes are still inefficient and prohibitive, especially for datasets of considerable size and complicated functions. For example, polynomial evaluation via FHE is prohibitive when the degree of polynomials is not a *very small constant* and the polynomial cannot be written in a batch-friendly form. In contrast, our protocol is efficient and will not be subject to these limitations.

2.4 Encryption Switching Protocol

The notion of encryption switching protocol was formalized by Couteau, Peters, and Pointcheval [6] in 2016 (see [3] for a more general construction). The encryption switching protocol applies to the scenario that two parties *secretly share* two private keys for a multiplicatively homomorphic encryption scheme and an

additively homomorphic encryption scheme, respectively. Both of them can individually encrypt a message, but neither of them can decrypt a ciphertext unless they cooperate to perform threshold decryption. This leads to a framework for two parties to cooperatively switch a ciphertext from one cryptosystem to the other and follow a deterministic computation path on the ciphertexts together until a computation result is reached. The encryption switching protocol also shows efficiency in practice [5]. Although we utilize this encryption switching idea in our work, we are considering a *different* scenario: one party holds the two complete private keys while the other holds only the corresponding public keys. In this scenario, given an encrypted value, the party holding the two private keys can decrypt this ciphertext and learn the corresponding plaintext herself, which leads to entirely different definitions and solutions from [6, 3].

2.5 Fairness Based on Blockchain

The blockchain paradigm is the underlying data structure, along with the emergence of Bitcoin [21]. It is deployed in a peer-to-peer (P2P) network, where all nodes follow a consensus mechanism. Ethereum [28] is the first platform that introduced blockchain-based smart contracts. After the deployment of a smart contract, nodes in the network execute the instructions specified by this smart contract and users. If the majority of nodes honestly follow the consensus mechanism, a blockchain can be deemed as a (semi-)honest third party with a *public* execution transcript.

It is known that general protocols in the absence of a third party cannot guarantee fairness when one of the parties is corrupted for 2PC problems [4]. Because of the emergence of the blockchain and smart contracts, some researchers recently integrated them into protocols as a third party to achieve fairness. A few of blockchain-based protocols, such as [2, 1, 16], ensure fairness of 2PC via a mechanism called *claim-or-refund*, in which a malicious party who aborts ahead of specified time will be forced to pay a monetary penalty.

For data trading, a few results based on the blockchain are also proposed to ensure data consistency through the claim-or-refund mechanism, such as [8, 10, 11]. These results mainly focus on data delivery, in which data sellers intend to sell an entire dataset or file. A few other results are proposed for collecting data from specified data generators, such as [19, 15, 20]. As a comparison, in our scenario, only two parties are involved, and the result evaluated on the function delegated by the buyer is delivered instead of the entire dataset.

3 Definitions and Generic Construction

According to earlier discussion in Section 1, we need a practical mechanism to capture the requirements of correctness, privacy, consistency and fairness for the data trading scenario. In this section, we model such a data trading scenario as a cryptographic notion called *blind polynomial evaluation*, aiming to resolve concerns of the first three requirements during data trading. The concern of fairness will be postponed to Section 6.

3.1 Blind Polynomial Evaluation

We consider the scenario that a seller Sarah initially uploads her entire dataset $\boldsymbol{x}$ in an encrypted form (denoted by $\boldsymbol{c_x}$) to a public place (*e.g.*, cloud) and publishes a hash value of $\boldsymbol{c_x}$ for consistency check. Then she can start trading data with potential buyers who have downloaded $\boldsymbol{c_x}$. We note that for Business-to-Customer (B2C) scenarios, the seller Sarah can indeed play the role of the storage server, and here we use *public place* to generalize the description that may also involve Customer-to-Customer (C2C) scenarios. It is typically cheap for individual sellers to use cloud storage service, and in this way sellers even do not need to store $\boldsymbol{c_x}$ locally. Since Sarah only needs to encrypt her dataset once, this approach avoids one-time use cost. We also note that once Sarah wants to update a portion of the dataset, she can only update this portion and the hash value. Moreover, since buyers can download the encrypted dataset without interaction with Sarah before a data trading, this approach, in some sense, largely saves Sarah's communication cost. Furthermore, the hash value of $\boldsymbol{c_x}$ ensures data consistency of transactions. Once the encrypted dataset is uploaded, Bob, as a potential buyer, can download $\boldsymbol{c_x}$ and check the hash value of $\boldsymbol{c_x}$.

Sarah, as a sender, in a blind fashion helps Bob, as a receiver, evaluates on the dataset $\boldsymbol{x}$ a function that belongs to the set $\mathcal{P}$, which contains all ℓ-variate polynomials with $k+1$ terms in the sparse representation of

$$P(\boldsymbol{x}) = \sum_{i=0}^{k} b_i \prod_{j=1}^{\ell} x_j^{d_{ij}} ,$$

where $\boldsymbol{x} \in (\mathbb{Z}_n^*)^\ell$, κ is the security parameter, $\ell, k \in \mathcal{O}(\kappa^c)$ for large enough constant $c > 0$, all $b_i, d_{ij} \in \mathcal{O}(2^\kappa)$, and $d_{0j} = 0$ for $j = 1 \dots \ell$. We denote the terms of $P(\boldsymbol{x})$ by $P_i(\boldsymbol{x}) = b_i \prod_{j=1}^{\ell} x_j^{d_{ij}}$ for $i = 1, \dots, k$ and $P_0(\boldsymbol{x}) = b_0$, such that $P(\boldsymbol{x}) = \sum_{i=0}^{k} P_i(\boldsymbol{x})$. Note that polynomials with the number of variates less than ℓ and number of terms less than $k+1$ can also be written in this form by simply setting certain b_i and d_{ij} to be 0. We call such a procedure *blind polynomial evaluation* and define its functionality of this notion as follows.

Definition 1. *The two-round functionality of the blind polynomial evaluation protocol between a receiver and a sender is presented below.*

1. (a) *The receiver inputs an encrypted vector* $\boldsymbol{c_x}$ *corresponding to a plaintext vector* $\boldsymbol{x}$ *and public information* pi. *The sender inputs a trapdoor* td *and public information* pi'.
   (b) *If* $\mathsf{pi} = \mathsf{pi}'$, td *is correct for* $\boldsymbol{c_x}$ *and* pi, *both parties receive* $\mathsf{continue}$ *and proceed to the next round. Otherwise, both parties receive* $\perp$ *with abortion.*
2. (a) *The receiver inputs the description of a polynomial* $P \in \mathcal{P}$.
   (b) *If* $P \in \mathcal{P}$, *the receiver receives the evaluation result* $P(\boldsymbol{x})$ *and the sender receives* $\mathsf{nothing}$. *Otherwise, both parties receive* $\perp$ *with abortion.*

We note that pi (resp. pi') includes public parameters such as public keys for $\boldsymbol{c_x}$. From this definition, the receiver only receives the result $P(\boldsymbol{x})$ at the end

without leaking any more information to the sender. Here, we allow buyer to set the polynomial $P(\boldsymbol{x}) = x_i$ to retrieve the i-th entry of $\boldsymbol{x}$. Our goal is for the seller to avoid revealing the entire dataset at one time, and it is acceptable that buyer retrieves the entire dataset via numerous transactions. If the scenario is to avoid leaking information of single entries, the dataset could simply be processed by differential privacy [9] techniques at the beginning.

As stated in Section 1.1, to support polynomial evaluation on encrypted datasets, and meanwhile, to make it *practical*, we utilize in our construction two *compatible* homomorphic encryption schemes $\Pi_\times$ and Π_+, where $\Pi_\times$ is multiplicatively homomorphic, and Π_+ is additively homomorphic. Here we call two schemes $\Pi_\times$ and Π_+ *compatible* if the plaintext space of Π_+ is a ring $\mathbb{R}$ and that of $\Pi_\times$ is a monoid $\mathbb{M}$ with $\mathbb{R} \cap \mathbb{M} = \mathbb{R}^*$, where $\mathbb{R}^*$ is the set of invertible elements of $\mathbb{R}$ [3]. The main idea of our construction is that Sarah first encrypts via $\Pi_\times$ all entries of the dataset, which then allows Bob to multiply encrypted values of the encrypted dataset directly. After multiplications, Sarah, in a *blind* manner, helps Bob switch all encrypted multiplication results to ciphertexts of Π_+, which again allows Bob to add encrypted values of these ciphertexts. Finally, we let Sarah once again in a *blind* fashion provide Bob with the final decrypted result.

To avoid duplicate definitions, in the rest of this section we denote the multiplicatively homomorphic encryption scheme as $\Pi_\times = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Mul})$ with a key pair $(\mathsf{pk}_\times, \mathsf{sk}_\times)$, message space $\mathcal{M}_\times$, and ciphertext space $\mathcal{C}_\times$, and the additively homomorphic encryption scheme as $\Pi_+ = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Add}, \mathsf{Mul})$ with a key pair $(\mathsf{pk}_+, \mathsf{sk}_+)$, message space $\mathcal{M}_+$, and ciphertext space $\mathcal{C}_+$. Here we require Π_+ to support efficient multiplication of an encrypted value and a constant. Since most additively homomorphic encryption schemes have this property, this requirement can be easily satisfied. We assume that $\Pi_\times$ and Π_+ are *compatible* and both IND-CPA (Indistinguishability under Chosen Plaintext Attack) secure. We write $r \leftarrow\!\!\$\, S$ for sampling r uniformly from a set S.

3.2 Definitions of Building Blocks

To formalize our idea, we introduce the following building blocks. We first recall the definition of twin-ciphertext pair.

Definition 2 (Twin-Ciphertext Pair [6]). *For two encryption schemes $\Pi_\times$ and Π_+, we call a pair of ciphertexts $(c_\times, c_+) \in (\mathcal{C}_\times, \mathcal{C}_+)$ a twin-ciphertext pair if $c_\times$ is an encryption of a message $m_\times$ under $\Pi_\times$, c_+ is an encryption of a message m_+ under Π_+, $m_\times, m_+ \in \mathcal{M}_\times \cap \mathcal{M}_+$ and $m_\times = m_+$.*

Then in Table 1, we present some languages for relations and their corresponding zero-knowledge ideal functionality. The functionality $\mathcal{F}_{\mathsf{zk}}^{\mathsf{TwinCtx}}$ is for proving that a ciphertext pair is a twin-ciphertext pair, $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EncValue}}$ is for proof of encrypted value for Π_+, and $\mathcal{F}_{\mathsf{zk}}^{\mathsf{sk}_+}$ is for proof of private key sk_+ of the scheme Π_+. Each ideal functionality receives the statement from both the prover and verifier, and a witness from the prover. It outputs accept to the verifier if the statement from both parties are the same and true, and reject otherwise.

Table 1. Languages for relations and their zero-knowledge ideal functionalities.

Language for relation	Functionality
$L_{\mathsf{TwinCtx}} = \{(c_\times \in \mathcal{C}_\times, c_+ \in \mathcal{C}_+, \mathsf{pk}_\times, \mathsf{pk}_+) \mid \exists(m_\times, m_+, r_+, \mathsf{sk}_\times), s.t.$ $\mathsf{sk}_\times$ is the private key of $\mathsf{pk}_\times \wedge m_\times = \Pi_\times.\mathsf{Dec}_{\mathsf{pk}_\times}(c_\times, \mathsf{sk}_\times)$ $\wedge\, c_+ = \Pi_+.\mathsf{Enc}_{\mathsf{pk}_+}(m_+; r_+) \wedge m_\times = m_+\}$	$\mathcal{F}_{\mathsf{zk}}^{\mathsf{TwinCtx}}$
$L_{\mathsf{EncValue}} = \{(c_+ \in \mathcal{C}_+, m_+ \in \mathcal{M}_+, \mathsf{pk}_+) \mid \exists(r_+), s.t.$ $c_+ = \Pi_+.\mathsf{Enc}_{\mathsf{pk}_+}(m_+; r_+)\}$	$\mathcal{F}_{\mathsf{zk}}^{\mathsf{EncValue}}$
$L_{\mathsf{sk}_+} = \{(\mathsf{pk}_+) \mid \exists(\mathsf{sk}_+), s.t. \mathsf{sk}_+$ is the private key of $\mathsf{pk}_+\}$	$\mathcal{F}_{\mathsf{zk}}^{\mathsf{sk}_+}$

3.3 Construction

We here present our generic construction of blind polynomial evaluation between the sender, *i.e.*, seller Sarah, and the receiver, *i.e.*, buyer Bob, in the $(\mathcal{F}_{\mathsf{zk}}^{\mathsf{TwinCtx}}, \mathcal{F}_{\mathsf{zk}}^{\mathsf{EncValue}}, \mathcal{F}_{\mathsf{zk}}^{\mathsf{sk}_+})$-hybrid model in Fig. 1.

Now we define the security of this construction. For the receiver's security, we should guarantee that: (i) A malicious sender cannot deviate from the protocol without being detected (with protocol abortion); (ii) The view of the sender can be simulated, *i.e.*, the sender *learns nothing*. The ideal functionalities indeed guarantee the first requirement in the hybrid model, that is, the sender must return the correct switched ciphertexts and the decryption result. We now define the receiver's security in the hybrid model against malicious senders as follows.

Definition 3 (Receiver's Security). *For all adversaries $\mathcal{A}$ running in probabilistic polynomial-time (*PPT*) with input* pi, $\mathsf{sk}_\times$, sk_+, *and auxiliary input z playing the sender's role in the $(\mathcal{F}_{\mathsf{zk}}^{\mathsf{TwinCtx}}, \mathcal{F}_{\mathsf{zk}}^{\mathsf{EncValue}}, \mathcal{F}_{\mathsf{zk}}^{\mathsf{sk}_+})$-hybrid model, there exists a* PPT *simulator $\mathcal{S}$ given* pi, $\mathsf{sk}_\times$, sk_+ *and z in the ideal model of blind polynomial evaluation, such that the output of $\mathcal{S}$ is (perfectly) indistinguishable from the view of $\mathcal{A}$.*

For the sender's security, it is necessary to ensure that after the evaluation of a polynomial P, the receiver cannot obtain more information than he should, *i.e.*, Bob only retrieves $P(\boldsymbol{x})$. In the hybrid model, a malicious Bob can only cause abortion or send different ciphertexts to Sarah instead of the ciphertexts according to an honest evaluation for $P(\boldsymbol{x})$. If Bob sends values in incorrect forms, he will be rejected and the protocol aborts. Since Bob obtains nothing, this behavior does not offend our security goal. If he sends ciphertexts that are different from an honest evaluation, but in the correct form, he indeed evaluates a different polynomial P' from P. We notice that if $\Pi_\times$ and Π_+ are both IND-CPA secure, Bob gains no advantage after he receives ciphertexts from Sarah (see more in Section 6.1). Hence, it is reasonable to let malicious Bob pick a polynomial P after seeing $\boldsymbol{c_x}$ and before the execution of the protocol, and then Bob behaves semi-honestly during the protocol. Hence, we define the adversary that plays the role of the receiver as $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$: $\mathcal{A}_0$ takes as input $\boldsymbol{c_x}$ and pi, and outputs the description of a polynomial $P \in \mathcal{P}$; $\mathcal{A}_1$ takes as input the description of P, $\boldsymbol{c_x}$ and pi, and acts as a semi-honest adversary to evaluate P on the encrypted

Inputs: The receiver Bob takes as input the description of a polynomial $P \in \mathcal{P}$, an encrypted vector $\boldsymbol{c_x}$ of $\Pi_\times$ corresponding to a vector $\boldsymbol{x}$. The sender Sarah takes as input the keys $\mathsf{sk}_\times$ and sk_+ (*i.e.*, the trapdoor td) for $\Pi_\times$ and Π_+, respectively. Both parties also take as input the public information pi containing $\mathsf{pk}_\times$, pk_+.

1. Sarah checks if keys $\mathsf{sk}_\times$ and sk_+ are correct, and aborts if they are incorrect. Bob, as the verifier, calls $\mathcal{F}_{\mathsf{zk}}^{\mathsf{sk}_+}$ with the prover Sarah for pk_+. If $\mathcal{F}_{\mathsf{zk}}^{\mathsf{sk}_+}$ outputs accept, Bob continues. Otherwise, Bob halts and outputs $\perp$.
2. Bob computes on $\boldsymbol{c_x}$ according to the term P_i to derive $c_{\times,P_i(\boldsymbol{x})}$ for $i = 1, \ldots, k$. If a coefficient $b_i = 0$, Bob picks a random ciphertext of $\Pi_\times$ as $c_{\times,P_i(\boldsymbol{x})}$.
3. Bob interacts with Sarah for all $c_{\times,P_i(\boldsymbol{x})}$'s, $i = 1, \ldots, k$, following the *encryption switching procedure* below to switch the underlying encryption scheme of these ciphertexts. At the end, Bob retrieves switched ciphertexts $c_{+,P_i(\boldsymbol{x})}$'s, $i = 1, \ldots, k$.
   (a) Bob picks $s \leftarrow\!\!\$\, \mathcal{M}_\times$ and computes $c_s \leftarrow \Pi_\times.\mathsf{Enc}_{\mathsf{pk}_\times}(s)$. Then Bob computes $c'_{\times,P_i(\boldsymbol{x})} \leftarrow \Pi_\times.\mathsf{Mul}_{\mathsf{pk}_\times}(c_{\times,P_i(\boldsymbol{x})}, c_s)$ to randomize $c_{\times,P_i(\boldsymbol{x})}$, and sends $c'_{\times,P_i(\boldsymbol{x})}$ to Sarah.
   (b) If $c'_{\times,P_i(\boldsymbol{x})} \in \mathcal{C}_\times$, Sarah decrypts $c'_{\times,P_i(\boldsymbol{x})}$, re-encrypts the decrypted value via Π_+ with pk_+ to derive $c'_{+,P_i(\boldsymbol{x})}$, and sends it to Bob. Otherwise, Sarah halts and outputs $\perp$.
   (c) Bob, as the verifier, calls $\mathcal{F}_{\mathsf{zk}}^{\mathsf{TwinCtx}}$ with the prover Sarah for $(c'_{\times,P_i(\boldsymbol{x})}, c'_{+,P_i(\boldsymbol{x})})$. If $\mathcal{F}_{\mathsf{zk}}^{\mathsf{TwinCtx}}$ outputs accept, Bob continues. Otherwise, Bob halts and outputs $\perp$.
   (d) Bob computes $c_{+,P_i(\boldsymbol{x})} \leftarrow \Pi_+.\mathsf{Mul}_{\mathsf{pk}_+}(c'_{+,P_i(\boldsymbol{x})}, s^{-1})$, where s^{-1} is the multiplicative inverse of s (such an inverse exists as $\Pi_\times$ and Π_+ are compatible).
4. Bob computes $c_{+,P_0(\boldsymbol{x})} \leftarrow \Pi_+.\mathsf{Enc}_{\mathsf{pk}_+}(b_0)$ and sums encrypted values of $c_{+,P_i(\boldsymbol{x})}$ for $i = 0, \ldots, k$ (except $b_i = 0$) to obtain $c_{+,P(\boldsymbol{x})}$, which is the encrypted $P(\boldsymbol{x})$.
5. Bob retrieves the final result $m = P(\boldsymbol{x})$ via the following *result retrieval procedure*.
   (a) Bob picks $s \leftarrow\!\!\$\, \mathcal{M}_+$ and encrypts it via $c_s \leftarrow \Pi_+.\mathsf{Enc}_{\mathsf{pk}_+}(s)$. Then Bob computes $c'_{+,P(\boldsymbol{x})} \leftarrow \Pi_+.\mathsf{Add}_{\mathsf{pk}_+}(c_{+,P(\boldsymbol{x})}, c_s)$ and sends it to Sarah.
   (b) If $c'_{+,P(\boldsymbol{x})} \in \mathcal{C}_+$, Sarah decrypts it and sends the decrypted value m' to Bob.
   (c) Bob, as the verifier, calls $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EncValue}}$ with the prover Sarah for $c'_{+,P(\boldsymbol{x})}$ and m'. If $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EncValue}}$ outputs accept, Bob continues. Otherwise, Bob halts and outputs $\perp$.
   (d) Bob outputs the decrypted result $m \leftarrow m' - s$.

Fig. 1. Generic construction with $\mathcal{F}_{\mathsf{zk}}^{\mathsf{TwinCtx}}$, $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EncValue}}$, and $\mathcal{F}_{\mathsf{zk}}^{\mathsf{sk}_+}$.

dataset $\boldsymbol{c_x}$ with the sender. We abuse the notion representation slightly and use P to represent the description of the polynomial P if the context is clear.

Definition 4 (Sender's Security). *For every* PPT *adversary* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ *in the* $(\mathcal{F}_{\mathsf{zk}}^{\mathsf{TwinCtx}}, \mathcal{F}_{\mathsf{zk}}^{\mathsf{EncValue}}, \mathcal{F}_{\mathsf{zk}}^{\mathsf{sk}_+})$*-hybrid model with input* $\boldsymbol{c_x}$, pi *and auxiliary input* z *playing receiver's role, once the description of a polynomial is output via* $P \leftarrow \mathcal{A}_0(\boldsymbol{c_x}, \mathsf{pi}, z)$*, there exists a* PPT *simulator* $\mathcal{S}$ *taking* $\boldsymbol{c_x}$, pi *and* P *as input*

in the ideal model, such that the view simulated by $\mathcal{S}$ and the view of the semi-honest $\mathcal{A}_1$ in the hybrid model taking $\boldsymbol{c_x}$, pi, and P as input are computationally indistinguishable.

For the above generic construction, we have the theorem as follows.

Theorem 1. *If $\Pi_\times$ and Π_+ are both* IND-CPA*, the generic construction in the $(\mathcal{F}_{\mathsf{zk}}^{\mathsf{TwinCtx}}, \mathcal{F}_{\mathsf{zk}}^{\mathsf{EncValue}}, \mathcal{F}_{\mathsf{zk}}^{\mathsf{sk}_+})$-hybrid model guarantees both receiver's and sender's security.*

Therefore, to guarantee both receiver's and sender's security, we should ensure that both $\Pi_\times$ and Π_+ are IND-CPA secure, and functionalities $\mathcal{F}_{\mathsf{zk}}^{\mathsf{TwinCtx}}$, $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EncValue}}$, and $\mathcal{F}_{\mathsf{zk}}^{\mathsf{sk}_+}$ are securely realized in the presence of *malicious* adversaries.

4 Instantiation of Blind Polynomial Evaluation over $\mathbb{Z}_n^*$

To instantiate the generic construction of blind polynomial evaluation, we utilize a variant of ElGamal encryption scheme from [6] with the plaintext space of $\mathbb{Z}_n^*$ and the Paillier encryption scheme from [23] with the plaintext space of $\mathbb{Z}_n$, where n is a strong RSA modulus having two distinct prime factors of the same length. It is easy to see that these two schemes are compatible. Given these two schemes, we then provide protocols that securely realize $\mathcal{F}_{\mathsf{zk}}^{\mathsf{TwinCtx}}$, $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EncValue}}$, and $\mathcal{F}_{\mathsf{zk}}^{\mathsf{sk}_+}$ in the presence of malicious adversaries, which immediately leads to a secure blind polynomial evaluation protocol based on the generic construction.

For the following description, let κ and t be the security parameters, and negl be a negligible function. Algorithms implicitly take as input 1^κ. Our instantiation in this paper relies on the following computational hardness assumptions.

- The Decisional Diffie-Hellman (DDH) assumption in a cyclic group $\mathbb{G} = \langle g \rangle$ of order $q \in \Theta(2^\kappa)$ is that for all PPT adversaries $\mathcal{A}$, we have

$$\Pr\left[\mathcal{A}(\mathbb{G}, q, g, g^x, g^y, z_b) = b : \begin{matrix} x, y \leftarrow_\$ \mathbb{Z}_q; z_0 = g^{xy}; \\ z_1 \leftarrow_\$ \mathbb{G}; b \leftarrow_\$ \{0,1\} \end{matrix}\right] \leq 1/2 + \mathsf{negl}(\kappa) \ .$$

- The Decisional Composite Residuosity (DCR) assumption in $\mathbb{Z}_{n^2}^*$, where $n \in \Theta(2^\kappa)$ is a strong RSA modulus, is that for all PPT adversaries $\mathcal{A}$, we have

$$\Pr\left[\mathcal{A}(n, z_b) = b : \begin{matrix} r \leftarrow_\$ \mathbb{Z}_n^*; z_0 = r^n \bmod n^2; \\ z_1 \leftarrow_\$ \mathbb{Z}_{n^2}^*; b \leftarrow_\$ \{0,1\} \end{matrix}\right] \leq 1/2 + \mathsf{negl}(\kappa) \ .$$

- The Quadratic Residuosity (QR) assumption in $\mathbb{Z}_n^*$, where $n \in \Theta(2^\kappa)$ is a strong RSA modulus, is that for all PPT adversaries $\mathcal{A}$, we have

$$\Pr\left[\mathcal{A}(n, z_b) = b : \begin{matrix} r \leftarrow_\$ \mathbb{Z}_n^*; z_0 \leftarrow r^2 \bmod n; \\ z_1 \leftarrow_\$ \mathbb{J}_n; b \leftarrow_\$ \{0,1\} \end{matrix}\right] \leq 1/2 + \mathsf{negl}(\kappa) \ ,$$

  where $\mathbb{J}_n$ is the set of all elements of $\mathbb{Z}_n^*$ whose *Jacobi symbols* are $+1$.

4.1 ElGamal Encryption over $\mathbb{Z}_n^*$

We slightly modify a variant of ElGamal encryption scheme $\mathbb{Z}_n^*$-EG over $\mathbb{Z}_n^*$ introduced in [6] and use it as the multiplicatively homomorphic encryption scheme $\Pi_\times$. The description of $\mathbb{Z}_n^*$-EG is given below.

Key Generation The key generation algorithm KGen takes as input the security parameter 1^κ and generates a strong RSA modulus $n = pq$ where $p, q \in \Theta(2^\kappa)$ are distinct randomly-chosen safe primes having the same length. Then the algorithm follows the procedure below:

1. Compute $g_0 \leftarrow_\$ \mathbb{Z}_n^*$, $g \leftarrow -g_0^2$ to obtain a generator of $\mathbb{J}_n$ of order $\lambda = \mathsf{lcm}(p-1, q-1)$. Here $\mathbb{J}_n$ is the set of all elements of $\mathbb{Z}_n^*$ whose *Jacobi symbols* are $+1$.
2. Compute $v = [p^{-1} \bmod q] \cdot p \bmod n$, such that $v \equiv 0 \bmod p$ and $v \equiv 1 \bmod q$, and $\chi \leftarrow (1-v) \cdot g^{t_p} + v \cdot g^{t_q} \bmod n$ for an even $t_p \leftarrow_\$ \mathbb{Z}_\lambda$ and an odd $t_q \leftarrow_\$ \mathbb{Z}_\lambda$. Compute θ, such that $g^{2\theta} = \chi^2$, based on the Chinese Remainder Theorem.
3. Pick $s \leftarrow_\$ \mathbb{Z}_\lambda$, and set $h \leftarrow g^s$. Note that such (s, h) are components of the private key and the public key in the generic ElGamal encryption scheme.
4. Output the public key $\mathsf{pk}_\times \leftarrow (n, g, \chi, h)$ and the private key $\mathsf{sk}_\times \leftarrow (s, \theta, p, q)$. Note that we can derive λ, v, t_p, t_q from $\mathsf{sk}_\times$.

Encryption The encryption algorithm Enc takes as input a message $m \in \mathbb{Z}_n^*$ and a public key $\mathsf{pk}_\times$, and encodes m in $\mathbb{J}_n$ via $(m_1, m_2) \leftarrow (g^a, \chi^{-a} m) \in \mathbb{J}_n^2$ for $a \leftarrow_\$ \{1, \ldots, \lfloor n/2 \rfloor\}$ that satisfies $J_n(m) = (-1)^a$. Here J_n is an algorithm to compute the Jacobi symbol of a given value. Then the algorithm computes $c_J \leftarrow \mathbb{J}_n\text{-EG.Enc}(m_2) = (c_0 = g^r, c_1 = m_2 h^r)$ for $r \leftarrow_\$ \{1, \ldots, \lfloor n/2 \rfloor\}$. Finally, the algorithm returns the ciphertext $c \leftarrow (c_J = (c_0, c_1), m_1)$.

Decryption The decryption algorithm Dec takes as input a ciphertext $c = (c_J = (c_0, c_1), m_1)$ and a key pair $(\mathsf{pk}_\times, \mathsf{sk}_\times)$, checks whether $J_n(c_1) = 1$ and outputs $\perp$ if it is not. If the check passes, the algorithm recovers m_2 via $m_2 \leftarrow \mathbb{J}_n\text{-EG.Dec}(c_J) = c_1 / c_0^s \bmod n$ and computes $m_0 \leftarrow (1-v) \cdot m_1^{t_p} + v \cdot m_1^{t_q} \bmod n$. Finally, the algorithm returns the message $m \leftarrow m_0 m_2 \bmod n$.

Multiplication The multiplication algorithm Mul takes as input two ciphertexts $c = (c_0, c_1, m_1)$ and $c' = (c_0', c_1', m_1')$, and outputs $c'' = (c_0 \cdot c_0', c_1 \cdot c_1', m_1 \cdot m_1') = (c_0'', c_1'', m_1'')$. Assume that m is the plaintext of c and m' is the plaintext of c'. We can easily verify that c'' is the ciphertext of $m \cdot m'$.

For simplicity, we may omit the parameters $\mathsf{pk}_\times$ and $\mathsf{sk}_\times$ from input parameters of the above algorithms in the setting of no confusion. We may also use $\mathbb{Z}_n^*\text{-EG.Enc}(m; r, a)$ to explicitly indicate the random coins (r, a) for encryption.

The correctness of $\mathbb{J}_n$-EG is the same as the generic ElGamal encryption scheme: $c_1 / c_0^s = (m_2 h^r) / g^{rs} = (m_2 g^{rs}) / g^{rs} = m_2$ in $\mathbb{J}_n$. For an isomorphism f from $\mathbb{Z}_n^*$ to $\mathbb{Z}_p^* \times \mathbb{Z}_q^*$: $f(x) = ([x \bmod p], [x \bmod q])$, it is easily verified that

$$\begin{aligned} m_0 &= (1-v) \cdot m_1^{t_p} + v \cdot m_1^{t_q} \bmod n \leftrightarrow (m_1^{t_p}, m_1^{t_q}) = (g^{a t_p}, g^{a t_q}) \\ &= (g^{t_p}, g^{t_q})^a \leftrightarrow \left((1-v) \cdot g^{t_p} + v \cdot g^{t_q} \bmod n\right)^a \bmod n = \chi^a \bmod n \end{aligned}$$

and $m_0 m_2 = \chi^a \chi^{-a} m \bmod n = m$.

In [6], the authors proved the above $\mathbb{Z}_n^*$-EG is IND-CPA secure under the DDH assumption in $\mathbb{J}_n$ and the QR assumption in $\mathbb{Z}_n^*$.

4.2 Paillier Encryption over $\mathbb{Z}_n$

We use as the additively homomorphic encryption scheme the Paillier encryption scheme [23] $\mathbb{Z}_n$-P, *i.e.*, $\Pi_+ = \mathbb{Z}_n$-P. Its description is as follows.

Key Generation The algorithm KGen takes as input a security parameter 1^κ, and generates a strong RSA modulus $n = pq$, where $p, q \in \Theta(2^\kappa)$ are randomly-chosen safe primes having the same length. Then the algorithm outputs a key pair $(\mathsf{pk}_+ = n, \mathsf{sk}_+ = (p, q))$. From sk_+, we can compute $\lambda \leftarrow \mathsf{lcm}(p-1, q-1)$ and $d \leftarrow [\lambda^{-1} \bmod n] \cdot \lambda \bmod n\lambda$. Note that the public key $\mathsf{pk}_+ = n$ is equal to n of the public key $\mathsf{pk}_\times$ of $\mathbb{Z}_n^*$-EG.

Encryption The algorithm Enc takes as input a message $m \in \mathbb{Z}_n$ and the public key pk_+, and outputs the ciphertext $c \leftarrow (1+n)^m r^n \bmod n^2$, where $r \leftarrow_{\$} \mathbb{Z}_n^*$.

Decryption The algorithm Dec takes as input a ciphertext c, the key pair $(\mathsf{pk}_+, \mathsf{sk}_+)$, and returns the plaintext $m = ([c^d \bmod n^2] - 1)/n$.

Addition The algorithm Add takes as input two ciphertexts c and c', and outputs $c'' = c \cdot c' \bmod n^2$. Assume that m is the plaintext of c and m' is the plaintext of c'. We can easily check that c'' is the ciphertext of $m + m'$.

Multiplication The scalar multiplication algorithm Mul takes as input a ciphertext c and a constant s, and outputs $c' = c^s \bmod n^2$. Note that computing a constant power of a ciphertext is equivalent to multiplying its encrypted value by this constant.

Randomness Extraction The algorithm $\mathsf{ExtractR}$ takes as input a ciphertext c and a key pair $(\mathsf{pk}_+, \mathsf{sk}_+)$. It first computes $m \leftarrow \mathbb{Z}_n$-$\mathsf{P}.\mathsf{Dec}_{\mathsf{pk}_+}(c, \mathsf{sk}_+)$ and $c_0 \leftarrow c \cdot (1+n)^{-m}$. Since p and q have the same length, we have $\gcd(\lambda, n) = 1$. Hence, the algorithm can find a value x, such that $n \cdot x \bmod \lambda = 1$. Finally, the algorithm outputs the random coin $r \leftarrow c_0^x \bmod n$.

Since $c^d \equiv (1+n)^{md} r^{nd} \equiv (1+n)^{m[\lambda^{-1} \bmod n] \cdot \lambda} r^{n[\lambda^{-1} \bmod n] \cdot \lambda} \equiv (1+n)^m \equiv 1 + mn \bmod n^2$, we can easily extract the message via $m \leftarrow ([c^d \bmod n^2] - 1)/n$. We remark that $\mathbb{Z}_n$-P is IND-CPA secure under the DCR assumption [23]. For simplicity, we may omit the parameters pk_+ and sk_+ from input parameters of the above algorithms when the setting is clear. We may also use $\mathbb{Z}_n$-$\mathsf{P}.\mathsf{Enc}(m; r)$ to explicitly indicate the random coins r for encryption.

4.3 Instantiation of Functionalities

We introduce how to securely realize $\mathcal{F}_{\mathsf{zk}}^{\mathsf{TwinCtx}}$, $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EncValue}}$, and $\mathcal{F}_{\mathsf{zk}}^{\mathsf{sk}_+}$ based on $\mathbb{Z}_n^*$-EG and $\mathbb{Z}_n$-P. We provide protocols that are *public-coin honest-verifier zero-knowledge proof of knowledge*. There are several approaches to compile such protocols to protocols against malicious verifiers with *low overhead*, such as the

Fiat-Shamir heuristic [12]. Note that we could simply use proof of factoring techniques, such as [24] to securely realize $\mathcal{F}_{\mathsf{zk}}^{\mathsf{sk}_+}$ for $\mathbb{Z}_n$-P.

We use ideas of [7] to realize $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EncValue}}$ in Fig. 2. Here the prover Sarah can use $\mathbb{Z}_n$-$\mathsf{P}.\mathsf{ExtractR}$ to extract the random coins r_+ of the ciphertext c_+.

Inputs: Both the prover P and the verifier V take as input $c_+ \in \mathbb{Z}_{n^2}^*$, $m_+ \in \mathbb{Z}_n$, $\mathsf{pk}_+ = n$. P also takes as input the witness $r_+ \in \mathbb{Z}_n^*$.

1. P picks $s \leftarrow_{\$} \mathbb{Z}_n^*$ and sends $a \leftarrow \mathbb{Z}_n$-$\mathsf{P}.\mathsf{Enc}_{\mathsf{pk}_+}(0; s) = s^n \bmod n^2$ to V.
2. V returns $e \leftarrow_{\$} \{0,1\}^t$ to P if $a \in \mathbb{Z}_{n^2}^*$. Otherwise, V outputs reject.
3. P computes and sends to V the value $z \leftarrow s r_+^e \bmod n$.
4. V computes $c' = c_+(1+n)^{-m_+} \bmod n^2$. V outputs accept if $z^n \equiv a c'^e \bmod n^2$ and c', a, and z are all relatively prime to n. Otherwise, V outputs reject.

Fig. 2. Protocol $\mathsf{EncValue}$ associated with $\mathbb{Z}_n$-P.

Proposition 1. *The protocol* $\mathsf{EncValue}$ *associated with* $\mathbb{Z}_n$-P *is public-coin honest-verifier zero-knowledge proof of knowledge.*

Before we provide the protocol for $\mathcal{F}_{\mathsf{zk}}^{\mathsf{TwinCtx}}$, we introduce a zero-knowledge ideal functionality $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EncOne}}$ associated with the language that a given ciphertext $c_\times$ encrypts 1 as follows:

$$L_{\mathsf{EncOne}} = \{(c_\times = ((c_0,c_1), m_1) \in (\mathbb{Z}_n^*)^3, \mathsf{pk}_\times = (n, g, \chi, h)) \mid \exists (s, \theta), s.t. \\ h = g^s \bmod n \wedge \chi^2 \equiv g^{2\theta} \bmod n \wedge c_1 = m_1^{-\theta} c_0^s \bmod n\}.$$

If the plaintext of $c_\times = ((c_0 = g^r, c_1 = m_2 h^r), m_1 = g^a)$ is 1, which is encoded by $(m_1, m_2) = (g^a, \chi^{-a}) \in \mathbb{J}_n^2$ for an even a, we should have $c_1 = \chi^{-a} h^r = g^{-\theta a} g^{sr} = m_1^{-\theta} c_0^s$. Hence, the protocol that could be used to realize $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EncOne}}$ is given in Fig. 3 and the proposition for its security is in the following.

Proposition 2. *The protocol* EncOne *associated with* $\mathbb{Z}_n^*$-EG *is complete, sound, and honest-verifier zero-knowledge.*

For $\mathcal{F}_{\mathsf{zk}}^{\mathsf{TwinCtx}}$, the prover Sarah proves to the verifier Bob that a given ciphertext pair is a twin-ciphertext pair. We separate the protocol realizing $\mathcal{F}_{\mathsf{zk}}^{\mathsf{TwinCtx}}$ into two phases: offline and online phases, to obtain a more practical protocol[4].

For the offline phase (Fig. 4), the prover Sarah possessing $\mathsf{sk}_\times$ first generates a random ciphertext pair (c_0, c_0'), such that $c_0 = \mathbb{Z}_n^*$-$\mathsf{EG}.\mathsf{Enc}(m_0) = (c_{0J} = (c_{00}, c_{01}), m_{01})$ and $c_0' = \mathbb{Z}_n$-$\mathsf{P}.\mathsf{Enc}(m_0'; r_0')$, where $m_0 = m_0'$. Then P sends it to the verifier Bob and convinces Bob that it is indeed a twin-ciphertext pair without revealing information about the plaintexts and the corresponding random coins. The generated (c_0, c_0') will then be used in the online phase of $\mathsf{TwinCtx}$.

[4] Such an approach is similar to that of [6]. However, their security goal indeed cannot be achieved since the random coins of the ElGamal encryption cannot be extracted and the group order is hidden. We overcome the security faults for our scenario.

Inputs: Both the prover P and the verifier V take as input $\mathsf{pk}_\times = (n, g, \chi, h))$ and $(c_\times = ((c_0, c_1), m_1) \in (\mathbb{Z}_n^*)^3$. P also takes as input the witness (s, θ).

1. P randomly picks $u, v \leftarrow_\$ \mathbb{Z}_n$, computes $d_1 \leftarrow g^{2u} \bmod n$, $d_2 \leftarrow g^v \bmod n$, $d_3 \leftarrow m_1^{-u} c_0^v \bmod n$, and sends d_1, d_2, and d_3 to V.
2. V randomly picks $e \leftarrow_\$ \{0,1\}^t$ and sends it to P if $d_1, d_2, d_3 \in (\mathbb{Z}_n^*)^3$. Otherwise, V outputs reject.
3. P computes $z_1 \leftarrow u + e\theta \bmod \lambda$, $z_2 \leftarrow v + es \bmod \lambda$, and sends it to V.
4. V checks $g^{2z_1} \equiv d_1(\chi^e)^2 \bmod n$, $g^{z_2} \equiv d_2 h^e \bmod n$, and $m_1^{-z_1} c_0^{z_2} \equiv d_3 c_1^e \bmod n$. V outputs accept if all equations hold, and reject otherwise.

Fig. 3. Protocol EncOne associated with $\mathbb{Z}_n^*$-EG.

Inputs: Both the prover P and the verifier V take as input $c_0 = (c_{0J} = (c_{00}, c_{01}), m_{01}) \in (\mathbb{Z}_n^*)^3$, $c_0' \in \mathbb{Z}_{n^2}^*$, $\mathsf{pk}_+ = n$, $\mathsf{pk}_\times = (n, g, \chi, h)$. P also takes as input $(s, \theta) \in \mathsf{sk}_\times$, $m_0 = m_0' \in \mathbb{Z}_n$ and $r_0' \in \mathbb{Z}_n^*$, such that $c_0' = \mathbb{Z}_n\text{-}\mathsf{P}.\mathsf{Enc}_{\mathsf{pk}}(m_0'; r_0')$.

1. P generates t random ciphertext pairs (c_i, c_i'), such that $c_i = \mathbb{Z}_n^*\text{-}\mathsf{EG}.\mathsf{Enc}(m_i; r_i, a_i) = (c_{iJ} = (c_{i0}, c_{i1}), m_{i1})$, $c_i' = \mathbb{Z}_n\text{-}\mathsf{P}.\mathsf{Enc}(m_i'; r_i')$, and $m_i = m_i'$, for $i = 1, \ldots, t$. Then P sends them to V.
2. V picks $e = e_1 \cdots e_t \leftarrow \{0,1\}^t$ and returns e to P if $c_i \in (\mathbb{Z}_n^*)^3$ and $c_i' \in \mathbb{Z}_{n^2}^*$. Otherwise, V outputs reject.
3. For $i = 1, \ldots, t$,
   - if $e_i = 0$, P sends to V the values m_i, r_i, a_i and r_i';
   - if $e_i = 1$, P computes $R_i \leftarrow m_0/m_i \bmod n$, and encodes it as $(R_{i1}, R_{i2}) \leftarrow (g^{a_{R_i}}, \chi^{-a_{R_i}} R_i) \in \mathbb{J}_n^2$ for $a_{R_i} \leftarrow_\$ \{1, \ldots, \lfloor n/2 \rfloor\}$, such that $J_n(R_i) = (-1)^{a_{R_i}}$. Then P computes $\rho_i' \leftarrow r_i'^{R_i} \cdot r_0^{-1} \bmod n$. Finally, P sends R_i, a_{R_i} to V.
4. For $i = 1, \ldots, t$,
   - if $e_i = 0$, V checks the validity of (c_i, c_i') via $c_i = \mathbb{Z}_n^*\text{-}\mathsf{EG}.\mathsf{Enc}(m_i; r_i, a_i)$ and $c_i' = \mathbb{Z}_n\text{-}\mathsf{P}.\mathsf{Enc}(m_i; r_i')$;
   - if $e_i = 1$, V reconstructs $(R_{i1}, R_{i2}) \leftarrow (g^{a_{R_i}}, \chi^{-a_{R_i}} R_i) \in \mathbb{J}_n^2$, computes $D_i \leftarrow ((c_{i0} \cdot (c_{00})^{-1}, c_{i1} \cdot R_{i2} \cdot (c_{01})^{-1}), m_{i1} \cdot R_{i1} \cdot (m_{01})^{-1}) \in \mathbb{J}_n^3$ and $D_i' \leftarrow c_i'^{R_i} \cdot (c_0')^{-1} \bmod n^2$. Then, P proves to V that $D_i = \mathbb{Z}_n^*\text{-}\mathsf{EG}.\mathsf{Enc}(1)$ and $D_i' = \mathbb{Z}_n\text{-}\mathsf{P}.\mathsf{Enc}(0; \rho_i')$ hold using $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EncOne}}$ and $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EncValue}}$.

   If all verifications are accepted, V outputs accept, and reject otherwise.

Fig. 4. Protocol TwinCtx for $\mathbb{Z}_n^*$-EG and $\mathbb{Z}_n$-P — offline Phase.

In the online phase (Fig. 5), Sarah proves a given ciphertext $(c_\times, c_+)$ is a twin-ciphertext pair using (c_0, c_0'), as required in $\mathcal{F}_{\mathsf{zk}}^{\mathsf{TwinCtx}}$.

Intuitively, since m' is a random message and both $\mathbb{Z}_n$-P and $\mathbb{Z}_n^*$-EG are IND-CPA secure, m and the random coins of $(c_\times, c_+)$ will be preserved if (c_0, c_0') is correctly generated in the offline phase. We have the following proposition.

Proposition 3. *The* TwinCtx *protocol associated with* $\mathbb{Z}_n^*$-EG *and* $\mathbb{Z}_n$-P *is public-coin honest-verifier zero-knowledge proof of knowledge.*

Inputs: Both the prover P and the verifier V take as input $c_\times = (c_J = (c_{\times 0}, c_{\times 1}), m_{\times 1}) \in (\mathbb{Z}_n^*)^3$, $c_+ \in \mathbb{Z}_{n^2}^*$, $\mathsf{pk}_+ = n$, and $\mathsf{pk}_\times = (n, g, \chi, h)$. They have input a *twin-ciphertext pair* (c_0, c_0'), where $c_0 = ((c_{00}, c_{01}), m_{01})$ and c_0' from the offline phase. P also takes as input $(s, \theta) \in \mathsf{sk}_\times$, $m \in \mathbb{Z}_n$ and $r_+ \in \mathbb{Z}_n^*$, such that $c_+ = \mathbb{Z}_n\text{-}\mathsf{P}.\mathsf{Enc}_{\mathsf{pk}}(m; r_+)$, and (m_0', r_0') from the offline phase.

1. P computes $R \leftarrow m/m_0'$, and encodes R as $(R_1, R_2) \leftarrow (g^{a_R}, \chi^{-a_R} R) \in \mathbb{J}_n^2$ for $a_R \leftarrow_\$ \{1, \ldots, \lfloor n/2 \rfloor\}$, such that $J_n(R) = (-1)^{a_R}$. P computes $\rho_+ \leftarrow r_0'^R \cdot r_+^{-1} \bmod n$, and sends R, a_R to V.
2. V reconstructs $(R_1, R_2) \leftarrow (g^{a_R}, \chi^{-a_R} R) \in \mathbb{J}_n^2$, computes $D_\times \leftarrow ((c_{00} \cdot (c_{\times 0})^{-1}, c_{01} \cdot R_2 \cdot (c_{\times 1})^{-1}), m_{01} \cdot R_1 \cdot (m_{\times 1})^{-1}) \in \mathbb{J}_n^2$ and $D_+ \leftarrow c_0'^R \cdot (c_+)^{-1} \bmod n^2$. P proves to V that $D_\times = \mathbb{Z}_n^*\text{-}\mathsf{EG}.\mathsf{Enc}_{\mathsf{pk}_\times}(1)$ and $D_+ = \mathbb{Z}_n\text{-}\mathsf{P}.\mathsf{Enc}_{\mathsf{pk}_+}(0; \rho_+)$ hold using $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EncOne}}$ and $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EncValue}}$. If they hold, V outputs accept, and reject otherwise.

Fig. 5. Protocol $\mathsf{TwinCtx}$ for $\mathbb{Z}_n^*$-EG and $\mathbb{Z}_n$-P — online Phase.

Here each execution of $\mathsf{TwinCtx}$ generates and consumes a (random) twin-ciphertext pair, which is not desirable. We now introduce how to improve the efficiency of the $\mathsf{TwinCtx}$ protocol using the idea in [6, 17]. We first recall the notion *multi-exponentiation with encrypted bases* (MEB). The zero-knowledge functionality $\mathcal{F}_{\mathsf{zk}}^{\mathsf{MEB}}$ is for the relation associated with the language below:

$$L_{\mathsf{MEB}} = \{(n, \{\omega_i\}_{i=1}^k \in \{0,1\}^{\kappa \cdot k}, C, \{c_i\}_{i=1}^k \in (\mathbb{Z}_{n^2})^{k+1} \mid \exists (r, \{m_i, r_i\}_{i=1}^k), s.t.$$
$$\forall i \in \{1, \ldots, k\} c_i = \mathbb{Z}_n\text{-}\mathsf{P}.\mathsf{Enc}(m_i; r_i) \wedge C = \mathbb{Z}_n\text{-}\mathsf{P}.\mathsf{Enc}(\prod_{i=1}^k m_i^{\omega_i}; r)\}.$$

A protocol to realize $\mathcal{F}_{\mathsf{zk}}^{\mathsf{MEB}}$ was proposed in [6], and then it was improved and its security was formally proved in [17]. We can use the 5-round protocol in [17] to batch the executions of the online phase of $\mathsf{TwinCtx}$. In this setting, the prover P wants to prove to the verifier V that all pairs of $(c_i, c_i')_{i=1,\ldots,k}$ where $c_i = \mathbb{Z}_n^*\text{-}\mathsf{EG}.\mathsf{Enc}(m_i; r_i, a_i)$ and $c_i' = \mathbb{Z}_n\text{-}\mathsf{P}.\mathsf{Enc}(m_i'; r_i')$ are all twin-ciphertext pairs, *i.e.*, $m_i = m_i'$ given only one random twin-ciphertext pair generated in the offline phase of $\mathsf{TwinCtx}$. Here the common reference string (CRS) contains the description of a pseudo-random generator (PRG). The procedure for batching the executions of the online phase of $\mathsf{TwinCtx}$ is as follows.

1. V sends $\omega \leftarrow_\$ \{0,1\}^\kappa$ to P.
2. Both parties use ω as a seed for PRG to generate $(\omega_i)_{i=1,\ldots,k}$. Then both parties take ω_ith power for each entry of c_i and add them together to obtain C, such that $C = \mathbb{Z}_n^*\text{-}\mathsf{EG}.\mathsf{Enc}(\prod_{i=1}^k m_i^{\omega_i}; \sum_{i=1}^k \omega_i r_i, \sum_{i=1}^k \omega_i a_i)$. P picks $\rho \leftarrow_\$ \mathbb{Z}_n^*$ and sends $C' \leftarrow \mathbb{Z}_n\text{-}\mathsf{P}.\mathsf{Enc}(\prod_{i=1}^k m_i'^{\omega_i}; \rho)$ to V.
3. P and V uses $\mathcal{F}_{\mathsf{zk}}^{\mathsf{MEB}}$ on $((\omega_i)_{i=1,\ldots,k}, C', (c_i')_{i=1,\ldots,k})$.
4. If $\mathcal{F}_{\mathsf{zk}}^{\mathsf{MEB}}$ returns accept, P and V perform the online phase of $\mathsf{TwinCtx}$ for (C, C'), and V outputs what $\mathsf{TwinCtx}$ outputs. Otherwise, V returns reject.

MEB proves that C' is indeed the ciphertext of $\prod_{i=1}^{k} m_i'^{\omega_i}$. If $\prod_{i=1}^{k} m_i^{\omega_i} = \prod_{i=1}^{k} m_i'^{\omega_i}$ for random $(\omega_i)_{i=1,\ldots,k}$, we have $m_i = m_i'$ for $i = 1, \ldots, k$ with an overwhelming probability. Here we note that the messages from P in Step 2 and Step 3 can be combined, and the two protocols, MEB [17] and TwinCtx, can be performed in parallel. We can further pack the procedure, such that the online phase and the offline phase of TwinCtx are executed simultaneously. More precisely, the online phase uses the generated random twin-ciphertext pair, and meanwhile the offline phase proves that this generated ciphertext pair is indeed a twin-ciphertext pair. Such an approach can reduce the number of rounds of the encryption switching procedure (Step 3 of the generic construction) to 6.

5 Extension from $\mathbb{Z}_n^*$ to $\mathbb{Z}_n$

The $\mathbb{Z}_n^*$-EG scheme encrypts values in $\mathbb{Z}_n^*$. However, in some scenarios, it would be nice if one can encrypt the element 0. We illustrate a method to extend the protocol from $\mathbb{Z}_n^*$ to $\mathbb{Z}_n$ in this section. We recall the definition of computational equality as follows for our further discussion.

Definition 5 (Computational Equality [6]). *For two finite sets S_1 and S_2 with cardinalities $|S_1|, |S_2| \in \Theta(\kappa^c)$ for large enough constant $c > 0$, we call them computationally equal if for every PPT adversary $\mathcal{A}$, we have*

$$\Pr[m \in S_1 \oplus S_2 : m \leftarrow \mathcal{A}(S_1, S_2)] \leq \mathsf{negl}(\kappa) \ ,$$

where $S_1 \oplus S_2$ denotes the symmetric difference of S_1 and S_2.

We claim that $\mathbb{Z}_n$ and $\mathbb{Z}_n^* \cup \{0\}$ is computationally equal, since if we can find a value $m \in \mathbb{Z}_n \oplus (\mathbb{Z}_n^* \cup \{0\})$, we can factor the RSA modulus n, which contradicts the assumption that it is computationally hard to factor n. Hence, we only need to include 0 in the $\mathbb{Z}_n^*$-EG plaintext space to extend from $\mathbb{Z}_n^*$ to $\mathbb{Z}_n$.

The ciphertext of the new encryption scheme $\mathbb{Z}_n$-EG is a tuple $C = (c, u)$, where u is called *zero indicator*. A messages m is encrypted as follows.

- If $m \neq 0$, we compute $c \leftarrow \mathbb{Z}_n^*\text{-EG.Enc}(m)$, $u \leftarrow \mathbb{Z}_n^*\text{-EG.Enc}(1)$.
- If $m = 0$, we compute $c \leftarrow \mathbb{Z}_n^*\text{-EG.Enc}(r)$ for $r \leftarrow_\$ \mathbb{Z}_n^*$, and $u \leftarrow \mathbb{Z}_n^*\text{-EG.Enc}(\bar{g})$, where $\bar{g} \in \mathbb{J}_n$ is a predefined fixed value for $\mathbb{Z}_n$-EG.

Multiplication of two encrypted values with tuples (c_1, u_1) and (c_2, u_2) is by doing element-wise multiplications of these tuples. If the multiplication involves an encrypted zero, the zero indicator encrypts a *non-one* value and the first entry c encrypts a *random* value. Otherwise, the zero indicator encrypts 1, and c encrypts the multiplication result. To decrypt a tuple (c, u), the decryption algorithm decrypts the zero indicator via $z \leftarrow \mathbb{Z}_n^*\text{-EG.Dec}_{\mathsf{pk}_\times}(u, \mathsf{sk}_\times)$. If $z \neq 1$, the algorithm outputs $m \leftarrow 0$. Otherwise, the algorithm outputs $m \leftarrow \mathbb{Z}_n^*\text{-EG.Dec}_{\mathsf{pk}_\times}(c, \mathsf{sk}_\times)$. Because $\mathbb{Z}_n$-EG is based on $\mathbb{Z}_n^*$-EG, it is obvious that $\mathbb{Z}_n$-EG is also IND-CPA secure. Otherwise, we can construct a distinguisher to

break the IND-CPA security of $\mathbb{Z}_n^*$-EG. For the zero indicator, we can further encrypt values $\bar{g} \in \mathbb{J}_n$ without encoding and thus obtain *shorter* ciphertexts.

The extension from $\mathbb{Z}_n^*$ to $\mathbb{Z}_n$ affects the encryption switching procedure in the generic construction (Step 3), and we now illustrate how to modify this procedure in Fig 6. Our goal is to switch a $\mathbb{Z}_n$-EG ciphertext (c, u) to a $\mathbb{Z}_n$-P

Inputs: The receiver Bob takes as input $\mathsf{pk}_\times$, pk_+, and $\mathsf{sk}_\times$. The sender Sarah takes as input $C = (c, u)$, $\mathsf{pk}_\times$, pk_+.

1. Bob switches the encryption scheme of c from $\mathbb{Z}_n^*$-EG to $\mathbb{Z}_n$-P according to Step 3 of the generic construction, and obtains c_+. Bob executes Step 2 – 4 of the generic construction to evaluate the Lagrange polynomial L mentioned above on the encrypted value of u, and obtains a $\mathbb{Z}_n$-P ciphertext v. Denote the plaintext of c_+ by a and the plaintext of v by b, *i.e.*, $b \in \{0, 1\}$.
2. Bob picks $\alpha, \beta \leftarrow_\$ \mathbb{Z}_n$ and computes ciphertexts $c_\alpha \leftarrow \mathbb{Z}_n\text{-P.}\mathsf{Enc}_{\mathsf{pk}_+}(\alpha)$, $c_\beta \leftarrow \mathbb{Z}_n\text{-P.}\mathsf{Enc}_{\mathsf{pk}_+}(\beta)$, $c_{a+\alpha} \leftarrow \mathbb{Z}_n\text{-P.}\mathsf{Add}_{\mathsf{pk}_+}(c_+, c_\alpha)$, $c_{b+\beta} \leftarrow \mathbb{Z}_n\text{-P.}\mathsf{Add}_{\mathsf{pk}_+}(v, c_\beta)$. Then Bob sends $c_{a+\alpha}$ and $c_{b+\beta}$ to Sarah. Meanwhile, Bob locally computes $c_{a\beta} \leftarrow \mathbb{Z}_n\text{-P.}\mathsf{Mul}_{\mathsf{pk}_+}(c_+, \beta)$, $c_{b\alpha} \leftarrow \mathbb{Z}_n\text{-P.}\mathsf{Mul}_{\mathsf{pk}_+}(v, \alpha)$, and $c_{\alpha\beta} \leftarrow \mathbb{Z}_n\text{-P.}\mathsf{Enc}_{\mathsf{pk}_+}(\alpha\beta)$.
3. Sarah extracts $a' \leftarrow \mathbb{Z}_n\text{-P.}\mathsf{Dec}_{\mathsf{pk}_+}(c_{a+\alpha}, \mathsf{sk}_+)$ and $b' \leftarrow \mathbb{Z}_n\text{-P.}\mathsf{Dec}_{\mathsf{pk}_+}(c_{b+\beta}, \mathsf{sk}_+)$, and the corresponding random coins via $\mathbb{Z}_n$-P.ExtractR. Then Sarah computes $c'_c \leftarrow \mathbb{Z}_n\text{-P.}\mathsf{Enc}_{\mathsf{pk}_+}(a' \cdot b')$, and sends c'_c to Bob. Sarah, as the prover, calls $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EncMul}}$ with Bob, as the verifier, for the multiplication relation of $c_{a+\alpha}$, $c_{b+\beta}$, and c'_c.
4. If the output of $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EncMul}}$ is accept, Bob outputs $c_c \leftarrow c'_c \cdot c_{a\beta}^{-1} \cdot c_{b\beta}^{-1} \cdot c_{\alpha\beta}^{-1} \bmod n^2$, which is the switched ciphertext of C. Otherwise, Bob outputs $\perp$ and halts.

Fig. 6. Encryption switching procedure from $\mathbb{Z}_n$-EG to $\mathbb{Z}_n$-P with $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EncMul}}$.

ciphertext. Let the maximum degree of the polynomial P be $d_{\max}$. For u inside (c, u), we know that u encrypts one value of $\{1, \bar{g}, \ldots, \bar{g}^{d_{\max}}\}$. We thus can construct a Lagrange polynomial L of (at most) degree $d_{\max}$, which maps 1 to 1 and values in $\{\bar{g}, \ldots, \bar{g}^{d_{\max}}\}$ to 0. If we evaluate L on the encrypted value of u, we derive encrypted 1 for ciphertexts of non-zero values and encrypted 0 for ciphertexts of zero according to the zero indicator. Then this encrypted evaluation result multiplied by the encrypted value of c leads to the switched ciphertext.

The procedure in Fig. 6 utilizes the zero-knowledge functionality $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EncMul}}$ for the multiplication of encrypted values relation associated with the language:

$$L_{\mathsf{EncMul}} = \{(c_a, c_b, c_c \in (\mathbb{Z}_{n^2}^*)^3, \mathsf{pk}_+ = n) \mid \exists (a, b, c, r_a, r_b, r_c), s.t. \\ ab \equiv c \bmod n \wedge \forall x \in \{a, b, c\},\ \mathbb{Z}_n\text{-P.}\mathsf{Enc}_{\mathsf{pk}_+}(x; r_x) = c_x\}.$$

The protocol that realizes $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EncMul}}$ can be found in [7]. Note that since α and β are random, $c'_{a+\alpha}$ and $c'_{b+\beta}$ do not leak any information about a, b, Sarah learns no information about C, c_+ and v during the protocol.

Similar to the $\mathbb{Z}_n^*$ case, we can pack the encryption switching procedure for the $\mathbb{Z}_n$ case. More precisely, Bob can switch for both c and terms of Lagrange polynomial L on u simultaneously via batching TwinCtx. After obtaining the switched ciphertexts (c_+ and v), Bob starts Step 3 of the encryption switching procedure from $\mathbb{Z}_n$-EG to $\mathbb{Z}_n$-P in parallel. Hence, we derive a 6-round procedure. We present an illustration of this procedure for switching one ciphertext in Fig. 7, and a very similar approach can be used to switch multiple ciphertexts.

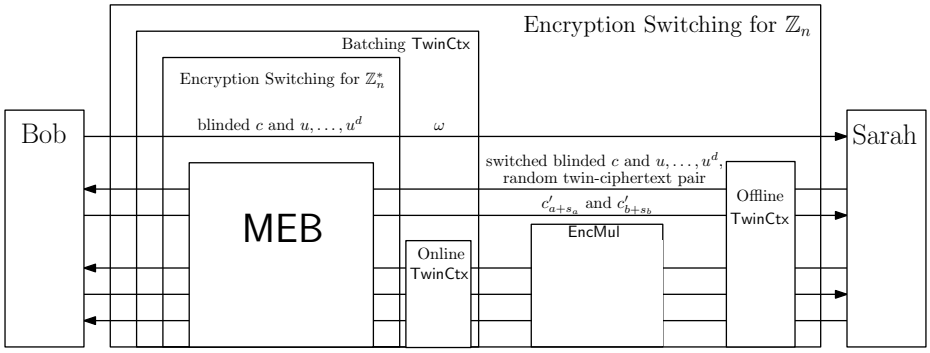


Fig. 7. Packing encryption switching procedure for the $\mathbb{Z}_n$ case.

6 Fair Exchange on Blockchain

In this section, we introduce how to achieve fairness via blockchain, such that the buyer Bob receives the evaluation result if and only if the seller Sarah gets paid from Bob. We first briefly introduce the underlying ideas.

We stress that Sarah has a negligible advantage to provide an incorrect result without being rejected if the blind polynomial evaluation protocol guarantees receivers' security. Meanwhile, Bob obtains no more information than the result $P(\boldsymbol{x})$ and cannot have any information about $P(\boldsymbol{x})$ before Step 5 of the generic construction. Therefore, we can compile the protocol EncValue associated with $\mathbb{Z}_n$-P via Fiat-Shamir heuristic to make it non-interactive, and deploy the proof verification process on smart contracts to achieve fair exchange. More precisely, Bob programs a smart contract, uploads parameters for EncValue, and freezes his payment on the contract. This smart contract receives Sarah's decrypted result, together with the proof, and verifies the proof. If the verification returns accept, Sarah will retrieve the payment automatically. Bob can remove the blind factor to obtain the final result. We call this approach *active* verification.

It is indeed possible to further reduce the cost. It is reasonable to assume that Sarah behaves mostly honestly, since Sarah may trade data many times with different buyers, and it will influence Sarah's credit if she is detected to behave dishonestly. We could require Sarah to pay a deposit on the smart contract

when she submits her proof. The smart contract now does not verify this proof. Alternatively, Bob verifies the proof off-chain, *i.e.*, Bob retrieves the proof from the smart contract and verifies it locally. If the proof is accepted, Sarah can retrieve the payment and her deposit after a specified period called *complaint period*. Hence, we save the cost of on-chain verification. Otherwise, Bob starts the verification procedure on the smart contract during the complaint period. If the smart contract indeed rejects the proof, it transfers the payment together with Sarah's deposit to Bob to penalize dishonest Sarah. Hence, if the latency of the complaint period is acceptable, this *passive* verification approach is cheaper. In what follows, we give a formal description and analysis for the ideas above.

6.1 Procedure Obliviousness

Before introducing the fair exchange protocol, we explicitly define for blind polynomial evaluation a security property called *procedure obliviousness*. Informally, the blind polynomial evaluation protocol achieves *procedure obliviousness* if the receiver of the protocol learns nothing beyond the public information $\boldsymbol{c_x}$ and pi before the *result retrieval procedure* of the generic construction. This property is to ensure that buyers must learn nothing if he aborts before the seller can claim the payment. The definition is given as follows.

Definition 6 (Procedure Obliviousness). *For every* PPT *adversary* $\mathcal{A}$ *with input* $\boldsymbol{c_x}$, pi *and auxiliary input* z *playing receiver's role until the beginning of the fifth step in the generic construction, there exists a* PPT *simulator* $\mathcal{S}$ *taking* $\boldsymbol{c_x}$, pi, z *as input in the ideal model, such that the view simulated by* $\mathcal{S}$ *is computationally indistinguishable from the view of* $\mathcal{A}$.

Our generic construction in Section 3 indeed achieves procedure obliviousness.

Theorem 2. *The generic construction in the hybrid model with the ideal functionalities* $\mathcal{F}_{\mathsf{zk}}^{\mathsf{TwinCtx}}$, $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EncValue}}$, *and* $\mathcal{F}_{\mathsf{zk}}^{\mathsf{sk}_+}$ *achieves procedure obliviousness.*

6.2 Security Requirements

Given that the blind polynomial evaluation protocol is procedure obliviousness, we proceed to introduce the fair exchange protocol for the result retrieval procedure of the generic construction over blockchain. Note that in this procedure, the seller Sarah will receive the blinded ciphertext $c'_{+,P(\boldsymbol{x})}$ from the buyer Bob. Now to achieve a fair exchange of the decryption result and the payment, we move the transfer of the decryption result for $c'_{+,P(\boldsymbol{x})}$ and the verification of $\mathsf{EncValue}$ protocol to blockchain. After the fair exchange, Bob can simply remove the blind factor to obtain the final result to finish the data trading procedure.

We first define the security requirements for the fair exchange of the data trading scenario between a data buyer and a data seller via *termination*, *buyer fairness*, and *seller fairness* in the following.

Termination. If at least one party is honest, the protocol will terminate, and all coins for the contract will be unlocked.

Buyer Fairness. An honest buyer is guaranteed that only if the seller provides the *correct* decryption result of the ciphertext c given by the buyer, the buyer will pay the seller p coins.

Seller Fairness. An honest seller is ensured that only if the buyer pays p coins to the seller, the buyer can learn the decryption result.

6.3 Protocol

We remark that our goal in this section is to integrate the transfer of the decryption result and the verification of $\mathsf{EncValue}$ protocol into the blockchain paradigm to achieve the security requirements defined in Section 6.2.

We first introduce how to compile the $\mathsf{EncValue}$ protocol for a Paillier ciphertext c, *i.e.*, $c'_{+,P(\boldsymbol{x})}$ in the generic construction, via the Fiat-Shamir heuristic to make it non-interactive and secure against malicious verifiers. Given a cryptographic hash function $\mathsf{H}\colon \{0,1\}^* \mapsto \{0,1\}^t$, the prover first picks $s \leftarrow_{\$} \mathbb{Z}_n^*$ and computes the value a as in $\mathsf{EncValue}$. Then the prover computes $e \leftarrow \mathsf{H}(n, c, m, a)$ to generate the challenge e. Finally, the prover computes z as in $\mathsf{EncValue}$ and sends (m, e, z) to verifiers. To verify the proof, verifiers compute $a \leftarrow z^n c'^{-e} \bmod n^2$ and output accept if and only if $e \leftarrow \mathsf{H}(n, c, m, a)$. Note that the size of the non-interactive proof is short to be deployed on blockchain, *e.g.*, 0.28125 KB for $||n|| = 2048$ and $t = 256$. The idea for the fair exchange of the evaluation result is to use a blockchain-enabled smart contract as a judge for the verification of this non-interactive proof when disputes happen.

The description of our fair exchange protocol basically follows the symbols and framework used in [10] (and also in [11]). As the same as [10], we abstract the communication by the synchronous communication model. This model assumes that the protocol is executed in rounds and all parties are aware of the current round. At the beginning of each round, parties receive all messages sent to them in the previous round. Meanwhile, all messages are sent within one round and received within the next round, *i.e.*, the communication is instantaneous.

We model the hash function (*e.g.*, `keccak 256`) used in the Fiat-Shamir heuristic via the global random oracle $\mathcal{H}$ and use the global ledger $\mathcal{L}$ (see [10] for more information) to model a blockchain (*e.g.*, Ethereum). Here we focus on the *passive verification* approach as described in Section 6. The ideal functionality $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$ is to model the blockchain-enabled smart contract $\mathcal{G}_{\mathsf{FairExchange}}$ with access to $\mathcal{L}$ and $\mathcal{H}$. Note that $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$, acting as a judge smart contract over the blockchain, interacts with $\mathcal{L}$, $\mathcal{H}$, the buyer Bob, and the seller Sarah. The description of $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$ is given in Fig. 8, and the description of the four-phase protocol for fair exchange between an honest buyer Bob and an honest seller Sarah is given in Fig. 9. In practice, because of the transparency of blockchain, both parties can check the code of the smart contract and start the protocol only if the smart contract correctly realizes the functionality of $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$.

In the initiation phase, $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$ receives from the buyer Bob the public key $\mathsf{pk}_+ = n$ for the Paillier encryption, the Paillier ciphertext c, as well as the

The ideal functionality $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$ locally stores addresses $addr_{\text{Bob}}$ and $addr_{\text{Sarah}}$ for both Bob and Sarah, respectively. It also maintains price p, state s, and the corresponding parameters for EncValue: n, c, m, e, and z.

Initiation

Upon receiving $(initiate, id, n, c, p)$ from the buyer Bob, store n, c, and p, and send $(freeze, id, \text{Bob}, p)$ to $\mathcal{L}$. If $\mathcal{L}$ responds with $(frozen, id, \text{Bob}, p)$, set $s = initialized$ and send $(initialized, id, n, c, p)$ to all parties.

Submission

Upon receiving $(submit, id, m, e, z)$ from the seller Sarah when $s = initialized$, send $(freeze, id, \text{Sarah}, p)$ to $\mathcal{L}$. If $\mathcal{L}$ responds with $(frozen, id, \text{Sarah}, p)$, set $s = submitted$, store m, e, and z, and send $(submitted, id, m, e, z)$ to all parties.
Otherwise, if no such a message from Sarah was received, send $(unfreeze, id, \text{Bob}, p)$ to $\mathcal{L}$ and abort.

Complaint

Upon receiving $(complain, id)$ from the buyer Bob when $s = submitted$, compute $c' \leftarrow c \cdot (1+n)^{-m} \bmod n^2$, $a \leftarrow z^n \cdot c'^{-e} \bmod n^2$, set $s = finalized$, and verify whether the equation $e = \mathcal{H}(n, c, m, a)$ holds.

- If the equation holds, send $(unfreeze, id, \text{Sarah}, 2p)$ to $\mathcal{L}$, $(sold, id)$ to all parties, and terminate.
- If the equation does not hold, send $(unfreeze, id, \text{Bob}, 2p)$ to $\mathcal{L}$, $(not\ sold, id)$ to all parties, and terminate.

If the message from Bob is $(finalize, id)$ when $s = submitted$, set $s = finalized$, send $(unfreeze, id, \text{Sarah}, 2p)$ to $\mathcal{L}$, output $(sold, id)$, and terminate.
Otherwise, if no such messages from Bob were received, proceed to the *payout* phase.

Payout

Upon receiving $(finalize, id)$ from Sarah when $s = submitted$, set $s = finalized$, send $(unfreeze, id, \text{Sarah}, 2p)$ to $\mathcal{L}$, output $(sold, id)$, and terminate.

Fig. 8. Ideal functionality $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$ for fair exchange smart contract.

price p for the evaluation result. Then $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$ locks p coins from Bob via $\mathcal{L}$ for the payment. The buyer Bob also sends n and c to the seller Sarah.

If the message from Bob and $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$ are consistent, Bob submits the decryption result and corresponding non-interactive zero-knowledge proof derived from EncValue to $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$ in the submission phase. Additionally, $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$ locks p coins from Bob via $\mathcal{L}$, which would be used to penalize dishonest Bob.

In the complaint phase, upon receiving the acknowledgment of Sarah's submitted message from $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$, Bob could locally run the verification of the non-interactive zero-knowledge proof. If the proof is incorrect, Bob needs to send the message to complain the dispute during the complaint phase *in time*. Once $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$ receives the complaint during the complaint phase, $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$ verifies the non-interactive zero-knowledge proof and resolves the dispute. If the verification is indeed incorrect, $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$ unlocks $2p$ coins to Bob (p coins sent

The description of the fair exchange protocol consists of the behavior of the honest buyer Bob and seller Sarah.

Initiation

Buyer Bob: Upon receiving input (buy, id, n, c, p), Bob sends (buy, id, n, c) to Sarah and $(initiate, id, n, c, p)$ to $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$. Then he proceeds to the *submission* phase.
Seller Sarah: Upon receiving input $(sell, id, n, c, m, r, p)$, Sarah proceeds to the *submission* phase.

Submission

Seller Sarah: Upon receiving (buy, id, n, c) from Bob, Sarah checks if she receives $(initialized, id, n, c, p)$ from $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$. If it is, Sarah computes $c' \leftarrow c \cdot (1+n)^{-m} \bmod n^2$, picks $s \leftarrow_{\$} \mathbb{Z}^*_n$, and computes $a = s^n \bmod n^2$, $e \leftarrow \mathcal{H}(n, c, m, a)$, and $z \leftarrow s \cdot r^e$. Then Sarah sends $(submit, id, m, e, z)$ to $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$.
If either message $(initialized, id, n, c, p)$ from $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$ or (buy, id, n, c) from Bob was not received, Sarah instead terminates the protocol.
Buyer Bob: Upon receiving $(submitted, id, m, e, z)$ from $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$, Bob proceeds to the *complaint* phase. If no message $(submitted, id, m, e, z)$ from $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$ was received, Bob terminates the protocol.

Complaint

Buyer Bob: Bob computes $c' \leftarrow c \cdot (1+n)^{-m} \bmod n^2$, $a \leftarrow z^n \cdot c'^{-e} \bmod n^2$. Bob then verifies whether $e = \mathcal{H}(n, c, m, a)$ holds. If it holds, Bob terminates the protocol by sending $(finalize, id)$ to $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$ and outputs $(bought, id, m)$. Otherwise, Bob sends $(complain, id)$ to $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$ and outputs $(not\ sold, id)$.
Seller Sarah: Upon receiving $(sold, id)$ or $(not\ sold, id)$ from $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$, Sarah outputs this message and terminates the protocol. Otherwise, if no such a message from $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$ was received, Sarah proceeds to the *payout* phase.

Payout

Seller Sarah: Sarah sends $(finalize, id)$ to $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$ and outputs $(sold, id)$.

Fig. 9. Protocol associated with $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$ between honest buyer Bob and seller Sarah.

back to honest Bob and p coins for penalizing dishonest Sarah sending incorrect proof). Otherwise, $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$ unlocks these $2p$ coins to Sarah. If the verification of the proof is accepted, Bob sends $(finalize, id)$ to finalize the fair exchange.

If no *complain* message or $finalize$ message from Bob is sent to $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$ during the complaint phase, Sarah sends $(finalize, id)$ to $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$ in the payout phase. Then $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$ unlocks the $2p$ coins to Sarah.

We note that smart contract for *active verification* approach mentioned in Section 6 is similar to $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$. The smart contract for active verification merges the submission, complaint, and payout phases together, *i.e.*, the smart contract verifies the proof once it receives the *submit* message. If the proof is correct, coins are sent to the seller. Otherwise, coins are sent back to the buyer.

6.4 Security Analysis

We now analyze the security requirements mentioned in Section 6.2 for the fair exchange protocol.

Termination The protocol always terminates, and all coins for the contract will be unlocked in one of the following cases when at least one parties act honestly.

No Abort. This case occurs when both parties act honestly, *i.e.*, Bob sends a *complain* message or $finalize$ message to $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$ in the complaint phase. According to the description of $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$, all coins will be unlocked at the end of the protocol.

Buyer Bob Aborts. After the initiation phase, Bob's abortion cannot stop the execution of $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$ when an honest Sarah involves. In case that Bob does not send $(finalize, id)$ in the complaint phase, Sarah could send $(finalize, id)$ to $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$ in the payout phase and coins will be sent to Sarah. Therefore, all coins will be unlocked at the end of the protocol.

Seller Sarah Aborts. This case occurs when Sarah does not submit decryption result and the corresponding non-interactive zero-knowledge proof in the submission phase. Here $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$ will ask $\mathcal{L}$ to unlock all p coins back to Bob and terminate the protocol.

Buyer Fairness The non-interactive zero-knowledge proof derived from $\mathsf{EncValue}$ guarantees the correctness of the decryption result for the ciphertext c. Note that a computationally bounded seller cannot provide a correct proof due to the security of $\mathsf{EncValue}$ compiled by the Fiat-Shamir heuristic under the random oracle model except a negligible probability.

Suppose that the decryption result m is incorrect, *i.e.*, the proof is incorrect. In that case, an honest buyer can complain to $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$ on time to prevent the payment and retrieve coins back. Since the verification of the proof on $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$ is the same as the verification executed by the honest buyer, coins will be back to the buyer. Thus, the seller here cannot retrieve the payment, and the protocol achieves buyer fairness.

Seller Fairness For seller fairness, we should ensure that once the buyer learns the decryption result of c, the honest seller should be paid. From the protocol, the buyer Bob learns the decryption result of c only if the honest seller Sarah submits it. For a submission of an honest seller, the buyer can choose to send *complain* or $finalize$ to $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$, or does nothing.

Suppose the buyer sends a *complain* message to $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$. In that case, the correct proof will still be accepted by $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$. Then the honest seller will receive the payment, together with her p coins frozen in the submission phase. For the $finalize$ message, $2p$ coins will be sent to the seller directly. If the buyer does nothing, the honest seller can send the message $finalize$ to $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$ to

retrieve the payment together with her own p coins frozen in the submission phase. Therefore, the protocol achieves seller fairness.

6.5 Possible Attacks and Countermeasures

We note that a malicious seller in practice may try to submit incorrect proof and hope that the buyer does not verify the proof or send *complain* on time. For this case, our solution is to penalize the malicious seller when her submitted proof is incorrect. In the submission phase, the seller is required to deposit p coins on $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$ when she submits the decryption result and the proof. Hence, if the proof is incorrect, the buyer can complain to $\mathcal{G}^{\mathcal{L},\mathcal{H}}_{\mathsf{FairExchange}}$ and retrieve these p coins to penalize the seller.

A malicious buyer may be able to perform a *Denial of Service (DoS)* attack. In a normal interaction of the data trading, the buyer Bob will perform computation on the encrypted dataset, run the *encryption switching procedure* with the seller Sarah, and execute the fair exchange protocol to retrieve the final decryption result. However, malicious Bob may perform a DoS attack by performing the encryption switching procedure with Sarah using garbage ciphertexts. These garbage ciphertexts are generated randomly rather than through computation on encrypted data. Then malicious Bob will always abort before the *result retrieval procedure*, *i.e.*, the fair exchange protocol. We note that the seller needs to conduct more computation than Bob during the $\mathsf{TwinCtx}$ protocol. Hence, if Bob launches this attack, though malicious Bob learns nothing from the protocol, it is unfair for Sarah to perform much more useless computation than Bob (since Bob here only generates garbage ciphertexts and acts as a verifier in $\mathsf{TwinCtx}$). This DoS attack cannot be avoided, but we still have countermeasures. A blacklist approach may be a possible solution, but alternatively, we provide another solution here. The solution is to let the buyer deploy the smart contract and freeze some coins on the contract before the data trading. These frozen coins can only be retrieved back at the end of the fair exchange, after a specified period, or directly treated as the payment. If the malicious buyer performs the DoS attack, he needs to pay the fee for the smart contract's deployment and freeze some coins on the smart contract at first, and thus we make it expensive to carry out such a DoS attack.

For the fair exchange protocol, if Sarah does not submit the decryption result and the corresponding proof, Bob is allowed to retrieve his coins frozen for the payment back. However, in practice, updates of blockchain follow a consensus mechanism, which allows malicious buyers to launch an attack based on this scenario of getting the payment back. In practice, it takes some time for the seller's submission to be confirmed on blockchain because of the consensus mechanism. At this point, after seeing the seller's submission, the malicious buyer can quickly submit a request to the smart contract to retrieve the frozen payment pretending that the seller has not submitted the decryption result. In this way, the malicious buyer's request may be confirmed by blockchain before the seller's submission. Thus, the malicious buyer gets the answer submitted by the seller while getting

back the frozen payment. Our solution to this attack is to set a time limit for the withdrawal of frozen payments in smart contracts. Within this period, the seller can submit the decryption result and proof, and the buyer is allowed to retrieve the frozen payment only after this period. Therefore, if the seller can submit the decryption result in time, malicious buyers' request to retrieve the frozen payment will not be accepted, so the attack cannot succeed.

7 Analysis

7.1 Round Complexity

We count the number of rounds of our two instantiations (both using the batch technique for TwinCtx) for protocols that are honest-verifier zero-knowledge (HVZK) or compiled by Fiat-Shamir heuristic. Since the offline phase of TwinCtx involves a cut-and-choose procedure, we do *not recommend* compiling this procedure via the Fiat-Shamir heuristic, and it is regarded as a *three-round* protocol even for the Fiat-Shamir heuristic. The total number of rounds of the protocol is equal to the number of rounds of the encryption switching procedure plus the number of rounds of the decryption procedure. For both the cases of $\mathbb{Z}_n^*$ and $\mathbb{Z}_n$, our instantiations only need 5 rounds under the Fiat-Shamir heuristic and 10 rounds under HVZK, which is very cheap for practical use.

7.2 Experimental Performance

We provide a *proof-of-concept* implementation to evaluate the performance of our blind polynomial evaluation protocol. The protocol is implemented in `C++` using the `NTL library` [25] for the underlying modular arithmetic on a single core of MacBook Air (2018) with a 1.6 GHz Intel©Core i5 CPU, 8GB of RAM, running macOS 10.15.4.

Table 2 provides the experimental performance of basic operations and ciphertext sizes in our instantiations. We give the total running time of per 10000 addition and multiplication operations, respectively. For the instantiation over $\mathbb{Z}_n$, we use zero indicators encrypting values of $\mathbb{J}_n$, as mentioned in Section 5.

Table 2. Experimental performance of basic operations and size of ciphertexts.

Ptx space	$\|\|n\|\|$	$10k \times$ Mul	$10k \times$ Add	ElGamal ctx	Paillier ctx
$\mathbb{Z}_n^*$	1024	0.2173s	0.1996s	0.375 KB	0.25 KB
$\mathbb{Z}_n$	1024	0.3834s	0.1971s	0.625 KB	0.25 KB
$\mathbb{Z}_n^*$	2048	0.6271s	0.6199s	0.750 KB	0.50 KB
$\mathbb{Z}_n$	2048	1.0387s	0.6143s	1.250 KB	0.50 KB

Table 3 presents the experimental performance of batching executions of TwinCtx and corresponding communication cost for security parameter $t = 32$.

We measure the running time for both the verifier and the prover when a random twin-ciphertext has been generated. The parameter k denotes the number of ciphertext pairs that are proved to be twin-ciphertext pairs. As the bottleneck of the protocol, its performance is efficient and practical.

Table 3. Performance of batching the executions of $\mathsf{TwinCtx}$ for $t = 32$.

$\|\|n\|\|$	k	Verifier	Prover	Communication cost
1024	128	0.6421s	4.6771s	172.50 KB
1024	256	1.1878s	9.2474s	316.81 KB
1024	512	2.3242s	18.5823s	605.44 KB
2048	128	4.2374s	28.7013s	344.44 KB
2048	256	8.1683s	58.3467s	632.75 KB
2048	512	16.1003s	117.2097s	1209.38 KB

7.3 Cost on Blockchain

We give a *proof-of-concept* implementation for the blockchain part illustrated in Section 6 by deploying it on a private network of Ethereum. We measure the computation cost via *gas consumption* of the smart contract execution, which only depends on the instructions executed by the Ethereum Virtual Machine. Table 4 presents the gas consumption and total transaction fee for both active and passive verification, with different security parameters κ.[5] Although our proof-of-concept implementation is not fully optimized, the gas consumption and fees are acceptable, especially for the passive verification approach.[6]

Acknowledgments. We thank the reviewers for their detailed and helpful comments. Y. Liu and Q. Wang were partially supported by the National Science Foundation of China under Grant No. 61672015 and Guangdong Provincial Key Laboratory (Grant No. 2020B121201001). Y. Liu and S.-M. Yiu were also partially supported by ITF, Hong Kong (ITS/173/18FP).

[5] Assume that the gas price is `10 Gwei` (a common price, albeit lower fees is possible). The total transaction fees (of US dollar) are calculated according to the *average price* of gas and coin on April 12th, 2020 (see more in https://etherscan.io/chart/gasprice). For the total fee, we take into account the total gas consumption of all functions for active verification and all functions except `complain` for passive verification.

[6] Note that since our implementation involves big integers and Ethereum today can only support integers represented by 256 bits, we have to use an external library. However, library instructions from therein will be pulled into the calling contract in the compilation. Hence, once a new version of Ethereum has better support of external library call, the cost of our protocol can further be *dramatically reduced*.

Table 4. Gas consumption of functions and total transaction fee for $t = 256$.

Mode	$\|\|n\|\|$	`initiate`	`submit`	`complain`	`getPaid`	Total Fee
Active	1024	620167	2807188	None	None	$5.41
Active	2048	1061626	13408995	None	None	$22.84
Passive	1024	635495	586386	2636802	30271	$1.98
Passive	2048	1076542	950408	13190574	30271	$3.25

References

1. Andrychowicz, M., Dziembowski, S., Malinowski, D., Mazurek, L.: Secure multiparty computations on bitcoin. In: 2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014. pp. 443–458. IEEE Computer Society (2014)
2. Bentov, I., Kumaresan, R.: How to use bitcoin to design fair protocols. In: Garay, J.A., Gennaro, R. (eds.) Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II. Lecture Notes in Computer Science, vol. 8617, pp. 421–439. Springer (2014)
3. Castagnos, G., Imbert, L., Laguillaumie, F.: Encryption switching protocols revisited: Switching modulo p. In: Katz, J., Shacham, H. (eds.) Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10401, pp. 255–287. Springer (2017)
4. Cleve, R.: Limits on the security of coin flips when half the processors are faulty (extended abstract). In: Hartmanis, J. (ed.) Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA. pp. 364–369. ACM (1986)
5. Couteau, G., Peters, T., Pointcheval, D.: Secure distributed computation on private inputs. In: García-Alfaro, J., Kranakis, E., Bonfante, G. (eds.) Foundations and Practice of Security - 8th International Symposium, FPS 2015, Clermont-Ferrand, France, October 26-28, 2015, Revised Selected Papers. Lecture Notes in Computer Science, vol. 9482, pp. 14–26. Springer (2015)
6. Couteau, G., Peters, T., Pointcheval, D.: Encryption switching protocols. In: Robshaw, M., Katz, J. (eds.) Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I. Lecture Notes in Computer Science, vol. 9814, pp. 308–338. Springer (2016)
7. Damgård, I., Jurik, M., Nielsen, J.B.: A generalization of paillier's public-key system with applications to electronic voting. Int. J. Inf. Sec. **9**(6), 371–385 (2010)
8. Delgado-Segura, S., Pérez-Solà, C., Navarro-Arribas, G., Herrera-Joancomartí, J.: A fair protocol for data trading based on bitcoin transactions. Future Generation Computer Systems (2017)
9. Dwork, C.: Differential privacy. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II. Lecture Notes in Computer Science, vol. 4052, pp. 1–12. Springer (2006)

10. Dziembowski, S., Eckey, L., Faust, S.: Fairswap: How to fairly exchange digital goods. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018. pp. 967–984. ACM (2018)
11. Eckey, L., Faust, S., Schlosser, B.: Optiswap: Fast optimistic fair exchange. In: Sun, H., Shieh, S., Gu, G., Ateniese, G. (eds.) ASIA CCS '20: The 15th ACM Asia Conference on Computer and Communications Security, Taipei, Taiwan, October 5-9, 2020. pp. 543–557. ACM (2020)
12. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings. Lecture Notes in Computer Science, vol. 263, pp. 186–194. Springer (1986)
13. Gamal, T.E.: A public key cryptosystem and a signature scheme based on discrete logarithms. In: Blakley, G.R., Chaum, D. (eds.) Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings. Lecture Notes in Computer Science, vol. 196, pp. 10–18. Springer (1984)
14. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Mitzenmacher, M. (ed.) Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009. pp. 169–178. ACM (2009)
15. Koutsos, V., Papadopoulos, D., Chatzopoulos, D., Tarkoma, S., Hui, P.: Agora: A privacy-aware data marketplace. In: 40th IEEE International Conference on Distributed Computing Systems, ICDCS 2020, Singapore, November 29 - December 1, 2020. pp. 1211–1212. IEEE (2020)
16. Kumaresan, R., Vaikuntanathan, V., Vasudevan, P.N.: Improvements to secure computation with penalties. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016. pp. 406–417. ACM (2016)
17. Liu, Y., Wang, Q., Yiu, S.: An improvement of multi-exponentiation with encrypted bases argument: Smaller and faster. In: Wu, Y., Yung, M. (eds.) Information Security and Cryptology - 16th International Conference, Inscrypt 2020, Guangzhou, China, December 11-14, 2020, Revised Selected Papers. Lecture Notes in Computer Science, vol. 12612, pp. 397–414. Springer (2020)
18. Liu, Y., Wang, Q., Yiu, S.: Blind polynomial evaluation and data trading. IACR Cryptol. ePrint Arch. **2021**, 413 (2021), https://eprint.iacr.org/2021/413
19. Lu, Y., Tang, Q., Wang, G.: Zebralancer: Private and anonymous crowdsourcing system atop open blockchain. In: 38th IEEE International Conference on Distributed Computing Systems, ICDCS 2018, Vienna, Austria, July 2-6, 2018. pp. 853–865. IEEE Computer Society (2018)
20. Lu, Y., Tang, Q., Wang, G.: Dragoon: Private decentralized hits made practical. In: 40th IEEE International Conference on Distributed Computing Systems, ICDCS 2020, Singapore, November 29 - December 1, 2020. pp. 910–920. IEEE (2020)
21. Nakamoto, S., et al.: Bitcoin: A peer-to-peer electronic cash system (2008)
22. Naor, M., Pinkas, B.: Oblivious polynomial evaluation. SIAM J. Comput. **35**(5), 1254–1281 (2006)
23. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques,

Prague, Czech Republic, May 2-6, 1999, Proceeding. Lecture Notes in Computer Science, vol. 1592, pp. 223–238. Springer (1999)
24. Poupard, G., Stern, J.: Short proofs of knowledge for factoring. In: Imai, H., Zheng, Y. (eds.) Public Key Cryptography, Third International Workshop on Practice and Theory in Public Key Cryptography, PKC 2000, Melbourne, Victoria, Australia, January 18-20, 2000, Proceedings. Lecture Notes in Computer Science, vol. 1751, pp. 147–166. Springer (2000)
25. Shoup, V.: Ntl: A library for doing number theory, http://www.shoup.net/ntl
26. Tassa, T., Jarrous, A., Ben-Ya'akov, Y.: Oblivious evaluation of multivariate polynomials. J. Mathematical Cryptology **7**(1), 1–29 (2013)
27. Valiant, L.G.: Universal circuits (preliminary report). In: Chandra, A.K., Wotschke, D., Friedman, E.P., Harrison, M.A. (eds.) Proceedings of the 8th Annual ACM Symposium on Theory of Computing, May 3-5, 1976, Hershey, Pennsylvania, USA. pp. 196–203. ACM (1976)
28. Wood, G., et al.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper **151**(2014), 1–32 (2014)
29. 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. IEEE Computer Society (1982)
30. Yao, A.C.: How to generate and exchange secrets (extended abstract). In: 27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986. pp. 162–167. IEEE Computer Society (1986)
31. Zhao, S., Yu, Y., Zhang, J., Liu, H.: Valiant's universal circuits revisited: An overall improvement and a lower bound. In: Galbraith, S.D., Moriai, S. (eds.) Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part I. Lecture Notes in Computer Science, vol. 11921, pp. 401–425. Springer (2019)

A Supplementary Material

A.1 Proof of Theorem 1

Theorem 1. *If $\Pi_\times$ and Π_+ are both* IND-CPA*, the generic construction in the $(\mathcal{F}_{\mathsf{zk}}^{\mathsf{TwinCtx}}, \mathcal{F}_{\mathsf{zk}}^{\mathsf{EncValue}}, \mathcal{F}_{\mathsf{zk}}^{\mathsf{sk}_+})$-hybrid model guarantees both receiver's and sender's security.*

Proof. We prove the security of the receiver and the sender, respectively, in the following.

Receiver's Security. We define a PPT simulator $\mathcal{S}$ as follows.

The simulator internally runs the adversary $\mathcal{A}$ playing the role of the sender, and feeds $\mathcal{A}$ with specified input private keys, pi and auxiliary input z. If the adversary $\mathcal{A}$ causes abortion during the execution, $\mathcal{S}$ sends abort to the ideal functionality of blind polynomial evaluation and outputs whatever $\mathcal{A}$ outputs.

When $\mathcal{A}$ sends sk_+ to $\mathcal{F}_{\mathsf{zk}}^{\mathsf{sk}_+}$, $\mathcal{S}$ does the same as what will be done by the ideal functionality $\mathcal{F}_{\mathsf{zk}}^{\mathsf{sk}_+}$ and the honest receiver. If the witness sk_+ is accepted, $\mathcal{S}$ records it.

For Step 3, $\mathcal{S}$ picks random ciphertext $c'_{\times,P_i(\boldsymbol{x})} \in \mathcal{C}_\times$ for $i = 1, \ldots, k$, and sends them to $\mathcal{A}$ as what will be done by an honest receiver. Then $\mathcal{S}$ receives switched ciphertexts from $\mathcal{A}$. When $\mathcal{A}$ sends witness for ciphertext pairs to $\mathcal{F}_{\mathsf{zk}}^{\mathsf{TwinCtx}}$, $\mathcal{S}$ does the same as what will be done by the ideal functionality $\mathcal{F}_{\mathsf{zk}}^{\mathsf{TwinCtx}}$ and the honest receiver. If all witnesses for $i = 1, \ldots, k$ are accepted, $\mathcal{S}$ records the private key $\mathsf{sk}_\times$.

For Step 4, $\mathcal{S}$ picks a random ciphertext $c'_{+,P(\boldsymbol{x})} \in \mathcal{C}_+$, and sends it to $\mathcal{A}$. Then $\mathcal{S}$ receives the result from $\mathcal{A}$. When $\mathcal{A}$ sends witness for $c'_{+,P(\boldsymbol{x})}$ to $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EncValue}}$, $\mathcal{S}$ does the same as what will be done by the ideal functionality $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EncValue}}$ and an honest receiver. If the witness is accepted, $\mathcal{S}$ sends $\mathsf{sk}_\times$ and sk_+ to the ideal functionality of blind polynomial evaluation, and outputs whatever $\mathcal{A}$ outputs to conclude the simulation. Since all ciphertexts received by $\mathcal{A}$ are all totally random in the simulation as well as in the real procedure, the output of $\mathcal{S}$ is perfectly indistinguishable from the view of $\mathcal{A}$, and the generic construction guarantees the receiver's security.

Sender's Security. We define a PPT simulator $\mathcal{S}$ as follows. After submitting pi, $c_{\boldsymbol{x}}$ to the ideal functionality of blind polynomial evaluation, If $\mathcal{S}$ receives $\perp$, $\mathcal{S}$ simply simulates the abortion due to the incorrect keys. Otherwise, $\mathcal{S}$ follows the simulation strategy below.

Recall that here $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$. Given input the description of the polynomial P from $P \leftarrow \mathcal{A}_0(c_{\boldsymbol{x}}, \mathsf{pi}, z)$, the simulator $\mathcal{S}$, as the receiver, sends P to the ideal functionality of blind polynomial evaluation and receives the evaluation result $P(\boldsymbol{x})$.

Then $\mathcal{S}$ follows the instructions of the generic construction to prepare accept messages for $\mathcal{F}_{\mathsf{zk}}^{\mathsf{sk}_+}$ and computes $c_{\times,P_i(\boldsymbol{x})}$ and $c'_{\times,P_i(\boldsymbol{x})}$ for $i = 1, \ldots, k$.

The simulator $\mathcal{S}$ encrypts $P(\boldsymbol{x})$ via Π_+ to obtain $c_{+,P(\boldsymbol{x})}$. If there exists an index $i^* \neq 0$, such that $b_{i^*} \neq 0$ for the polynomial P, we let $c_{+,P_i(\boldsymbol{x})}$ be a

random ciphertext that encrypts 0 for all $i \neq i^*$ and $i \neq 0$. The simulator $\mathcal{S}$ then generates $c_{+,P_0(\boldsymbol{x})}$ following the protocol. We let $c_{+,P_{i^*}(\boldsymbol{x})}$ be the ciphertext that the summation of all $c_{+,P_i(\boldsymbol{x})}$ (except $b_i = 0$) is equal to $c_{+,P(\boldsymbol{x})}$. Otherwise, we let $c_{+,P_i(\boldsymbol{x})}$ be a random ciphertext that encrypts 0 for all $i \neq 0$ and $c_{+,P_0(\boldsymbol{x})} = c_{+,P(\boldsymbol{x})}$. Then $\mathcal{S}$ computes $c'_{+,P_i(\boldsymbol{x})}$'s using the same blind factor s's as $c'_{\times,P_i(\boldsymbol{x})}$'s and uses accept as the outputs of $\mathcal{F}_{\mathsf{zk}}^{\mathsf{TwinCtx}}$.

For the result retrieval procedure, $\mathcal{S}$ follows the instructions of the generic construction to compute $c'_{+,P(\boldsymbol{x})}$ using a random blind factor s. $\mathcal{S}$ prepares $P(\boldsymbol{x})+s$ as the response from the sender when receiving $c'_{+,P(\boldsymbol{x})}$, and uses accept as the output of $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EncValue}}$. Finally, $\mathcal{S}$ sets $P(\boldsymbol{x})$ as the final result.

We note that the difference between the real execution and the simulation is the distribution of $c_{+,P_i(\boldsymbol{x})}$'s and $c'_{+,P_i(\boldsymbol{x})}$'s. Note that here $c'_{+,P_i(\boldsymbol{x})}$'s totally depend on $c_{+,P_i(\boldsymbol{x})}$'s. We analyze for the case that i^* exists. Security proof for the case that i^* does not exist follows similarly. We define the following sequence of games.

Game 0**.** This game is the transcript of the real execution in the hybrid model. All $c_{+,P_i(\boldsymbol{x})}$'s are computed as in a real execution of the generic construction.

Game i **(**$0 < i < i^*$**).** In this game, the first i ciphertexts are replaced by the ciphertexts computed as in the simulation, *i.e.*, for $j = 1, \ldots, i$, $c_{+,P_j(\boldsymbol{x})}$ is a random ciphertext that encrypts 0. The i^*-th ciphertext is also computed as in the simulation. The last $k - i$ (except the index i^*) ciphertexts $c_{+,P_i(\boldsymbol{x})}$'s are computed the same as the real execution.

Game i^***.** This game is the same as **Game** $i^* - 1$.

Game i **(**$i^* < i < k$**).** In this game, the first i ciphertexts are the ciphertexts computed as in the simulation. The last $k-i$ ciphertexts $c_{+,P_i(\boldsymbol{x})}$'s are computed the same as the real execution.

Game k**.** This game is the simulation.

We note that other values in the transcripts of this sequence of games are the same or totally depend on ciphertexts $c_{+,P_i(\boldsymbol{x})}$'s. If there exists a distinguisher $\mathcal{D}$ that can distinguish **Game** 0 and **Game** k, there should exist an index $i \in \{0, \ldots, k-1\}$, such that it is possible to distinguish **Game** i and **Game** $i+1$. The difference between **Game** i and **Game** $i+1$ is whether the ciphertext $c_{+,P_{i+1}(\boldsymbol{x})}$ encrypts the value as in the real execution or 0. However, since Π_+ is IND-CPA secure, **Game** i and **Game** $i+1$ for all $i \in \{0, \ldots, k-1\}$ are computationally indistinguishable. Thus, we have that **Game** 0 and **Game** k are computationally indistinguishable. Therefore, the view simulated by $\mathcal{S}$ is computationally indistinguishable from the view of the semi-honest $\mathcal{A}_1$ in the hybrid model, and the generic construction guarantees the sender's security. □

A.2 Proof of Proposition 1

Proposition 1. *The protocol* $\mathsf{EncValue}$ *associated with* $\mathbb{Z}_n$-P *is a public-coin honest-verifier zero-knowledge proof of knowledge.*

Proof. For completeness, it is easy to verify that $z^n \bmod n^2 = (sr_+^e)^n \bmod n^2 = s^n r_+^{ne} \bmod n^2 = ac'^e \bmod n^2$.

For special soundness, given any c and a pair of transcripts (a, e, z), (a, e', z') where $e \neq e'$, we have $z^n c'^{-e} \equiv z'^n c'^{-e'} \bmod n^2$. Rewriting this equation, we derive $(z/z')^n \equiv c'^{e-e'} \bmod n^2$. Since $e - e'$ is relatively prime to n except with a negligible probability, we can easily compute α and β, such that $\alpha n + \beta(e - e') = 1$. Let $\bar{c'} = c' \bmod n$ and $v = \bar{c'}^{\alpha}(z/z')^{\beta} \bmod n$, we have

$$\begin{aligned} v^n &\equiv \mathbb{Z}_n\text{-}\mathsf{P.Enc}(0; v) \equiv (\mathbb{Z}_n\text{-}\mathsf{P.Enc}(0, \bar{c'}))^{\alpha} \cdot (\mathbb{Z}_n\text{-}\mathsf{P.Enc}(0; z/z'))^{\beta} \\ &\equiv \bar{c'}^{\alpha n} \cdot (z/z')^{n\beta} \equiv c'^{\alpha n} \cdot c^{\beta(e-e')} \equiv c' \bmod n^2 . \end{aligned}$$

Therefore, v is an n-th root of c', and c' encrypts m_+.

For the special honest-verifier zero-knowledge property, a PPT simulator $\mathcal{S}$ can generate a transcript by randomly picking $z \leftarrow_{\$} \mathbb{Z}_n^*$, $e \leftarrow_{\$} \{0,1\}^t$ and computing $a \leftarrow z^n c'^{-e} \bmod n^2$. It is easy to see that the generated transcript (a, e, z) is perfectly indistinguishable from a real interaction transcript. □

A.3 Proof of Proposition 2

Proposition 2. *The protocol* EncOne *associated with* $\mathbb{Z}_n^*$-EG *is complete, sound, and honest-verifier zero-knowledge.*

Proof. For completeness of the protocol, it is easy to verify that $g^{2z_1} \equiv g^{2(u+e\theta)} \equiv g^{2u} g^{2\theta e} \equiv d_1(\chi^e)^2 \bmod n$, $g^{z_2} \equiv g^{v+es} \equiv g^v g^{se} \equiv d_2 h^e \bmod n$, and $m_1^{-z_1} c_0^{z_2} \equiv m_1^{-u-e\theta} c_0^{v+es} \equiv m_1^{-u} m_1^{-\theta e} c_0^v c_0^{se} \equiv d_3 c_1^e \bmod n$.

For soundness, given transcripts $(d_1, d_2, d_3, e, z_1, z_2)$ and $(d_1, d_2, d_3, e', z_1', z_2')$ with $e \neq e'$, we have $g^{2z_1}\chi^{-2e} \equiv g^{2z_1'}\chi^{-2e'} \bmod n$, $g^{z_2} h^{-e} \equiv g^{z_2'} h^{-e'} \bmod n$, and $m_1^{-z_1} c_0^{z_2} c_1^{-e} \equiv m_1^{-z_1'} c_0^{z_2'} c_1^{-e'} \bmod n$. Since $e - e'$ is relatively prime to λ except with a negligible probability, from these equations, we derive the following three equations: $g^{2(z_1 - z_1')/(e-e')} \equiv \chi^2 \bmod n$, $g^{(z_2 - z_2')/(e-e')} \equiv h \bmod n$, and $m_1^{(z_1' - z_1)/(e-e')} c_0^{(z_2 - z_2')/(e-e')} \equiv c_1 \bmod n$. Therefore, there exists $s = (z_2 - z_2')/(e - e')$ and $\theta = (z_1 - z_1')/(e - e')$, such that $h = g^s \bmod n$, $\chi^2 \equiv g^{2\theta} \bmod n$, $c_1 = m_1^{-\theta} c_0^s \bmod n$ hold, which implies that the protocol is sound.

For honest-verifier zero-knowledge property, a PPT simulator $\mathcal{S}$ can generate a transcript by randomly picking $z_1, z_2 \leftarrow_{\$} (\{0, \ldots, n/2\})^2$, $e \leftarrow_{\$} \{0,1\}^t$ and computing $d_1 \leftarrow g^{2z_1}\chi^{-2e} \bmod n$, $d_2 \leftarrow g^{z_1} h^{-e} \bmod n$, and $d_3 \leftarrow m_1^{-z_1} c_0^{z_2} c_1^{-e} \bmod n$. It is easy to see that the generated transcript $(d_1, d_2, d_3, e, z_1, z_2)$ is statistically indistinguishable from a real interaction transcript. □

Note that we can perform $\mathcal{F}_{\mathsf{zk}}^{\mathsf{sk}_+}$ for $\mathbb{Z}_n$-P (*e.g.*, [24]) in parallel, such that the simulator in the proof of security is able to obtain the group order λ. Thus, the protocol acquires the proof of knowledge property.

A.4 Proof of Proposition 3

Proposition 3. *The* $\mathsf{TwinCtx}$ *protocol associated with* $\mathbb{Z}_n^*$-EG *and* $\mathbb{Z}_n$-P *is public-coin honest-verifier zero-knowledge proof of knowledge.*

Proof. It is easy to verify the completeness of the protocol.

Let us first simulate the online phase of the protocol for a given twin-ciphertext pair $(c_\times, c_+)$, where $c_\times = (c_{\times 0}, c_{\times 1}, m_{\times 1})$. The simulator $\mathcal{S}$ picks randomly $R \leftarrow_\$ \mathbb{Z}_n^*$, and computes $c_0' \leftarrow c_+^{[R^{-1} \bmod n]} \cdot \tau^n \bmod n^2$ for $\tau \leftarrow_\$ \mathbb{Z}_n^*$, such that $D_+ = c_0'^R \cdot c_+^{-1}$ encrypts 0. $\mathcal{S}$ encodes R as $(R_1, R_2) \leftarrow (g^{a_R}, \chi^{-a_R} R) \in \mathbb{J}_n^2$ for $a_R \leftarrow_\$ \{1, \ldots, \lfloor n/2 \rfloor\}$, such that $J_n(R) = (-1)^{a_R}$. $\mathcal{S}$ then generates a random ciphertext $D_\times = (d_0, d_1, \tilde{m}_1)$ that encrypts 1, and computes $c_0 = (c_{00}, c_{01}, m_{01})$, where $c_{00} = d_0 c_{\times 0} \bmod n$, $c_{01} = d_1 c_{\times 1} R_2^{-1} \bmod n$, and $m_{01} = \tilde{m}_1 m_{\times 1} R_1^{-1} \bmod n$. The ciphertext pair (c_0, c_0') is the random twin-ciphertext pair for the online phase. It is easy to check that these simulated ciphertexts are perfectly distinguishable from a real execution. Then $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{zk}}^{sfEncOne}$ and $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EncValue}}$ for $D_\times$ and D_+.

Now given the twin-ciphertext pair (c_0, c_0'), where $c_0 = ((c_{00}, c_{01}), m_{01})$, and the challenge e, $\mathcal{S}$ simulates the offline phase of the protocol as follows. For every $e_i = 0$, the simulator $\mathcal{S}$ picks a random twin-ciphertext pair (c_i, c_i'), whose openings are now known by $\mathcal{S}$ in Step 3. For every $e_i = 1$, $\mathcal{S}$ picks randomly $R_i \leftarrow_\$ \mathbb{Z}_n^*$, and computes $c_i' \leftarrow c_0'^{[R_i^{-1} \bmod n]} \cdot \tau_i^n \bmod n^2$ for $\tau_i \leftarrow_\$ \mathbb{Z}_n^*$, such that $D_i' = c_i'^{R_i} \cdot (c_0')^{-1}$ encrypts 0. $\mathcal{S}$ encodes R_i as $(R_{i1}, R_{i2}) \leftarrow (g^{a_{R_i}}, \chi^{-a_{R_i}} R_i) \in \mathbb{J}_n^2$ for $a_{R_i} \leftarrow_\$ \{1, \ldots, \lfloor n/2 \rfloor\}$, such that $J_n(R_i) = (-1)^{a_{R_i}}$. $\mathcal{S}$ then generates a random ciphertext $D_i = (d_{i0}, d_{i1}, \tilde{m}_{i1})$ that encrypts 1, and computes $c_i = (c_{i0}, c_{i1}, m_{i1})$, where $c_{i0} = d_{i0} c_{00} \bmod n$, $c_{i1} = d_{i1} c_{01} R_{i2}^{-1} \bmod n$, and $m_{i1} = \tilde{m}_{i1} m_{01} R_{i1}^{-1} \bmod n$. Finally, $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EncOne}}$ and $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EncValue}}$ to conclude the simulation. It is easy to check that these simulated ciphertexts are perfectly distinguishable from those of a real execution.

Finally, we will show that the protocol has witness-extended emulation. The witness-extended emulator runs the protocol with a random challenge e for the offline phase. If the proof is accepted, the emulator rewinds to the second move until it gets another accepted transcript for a challenge $e' \neq e$. On average, the emulator will be making 2 transcripts, so it runs in expected polynomial time.

Since $e' \neq e$, there exists at least one pair $e_j \neq e_j'$. Now the emulator has the openings of the *twin-ciphertext pair* (c_j, c_j'). We can obtain the opening of D_j' from $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EncValue}}$ and (s, θ) from $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EncOne}}$. Thus, the emulator can compute the openings of c_0' from the openings of c_j' and D_j'. It is easy to verify that if (c_0, c_0') is not a twin-ciphertext pair, at least one check for challenges (e_j, e_j') will be rejected. Hence, we have all private inputs of P for the online phase. Now, using a similar technique as above, the emulator can extract the witness of the ciphertext pair $(c_\times, c_+)$ for the relation R_{TwinCtx}. Thus, the protocol has witness-extended emulation, and the soundness of the protocol follows. □

A.5 Proof of Theorem 2

Theorem 2. *The generic construction in the hybrid model with the ideal functionalities $\mathcal{F}_{\mathsf{zk}}^{\mathsf{TwinCtx}}$, $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EncValue}}$, and $\mathcal{F}_{\mathsf{zk}}^{\mathsf{sk}_+}$ achieves procedure obliviousness.*

Proof. We define a PPT simulator $\mathcal{S}$ that internally runs $\mathcal{A}$. After submitting pi, $c_{\boldsymbol{x}}$ to the ideal functionality of blind polynomial evaluation, if $\mathcal{S}$ receives $\perp$,

$\mathcal{S}$ simply simulates the abortion due to the incorrect keys. Otherwise, $\mathcal{S}$ follows the simulation strategy below.

Upon receiving the call for $\mathcal{F}_{\mathsf{zk}}^{\mathsf{sk}_+}$ from $\mathcal{A}$, $\mathcal{S}$ plays the role of $\mathcal{F}_{\mathsf{zk}}^{\mathsf{sk}_+}$ and replies accept to $\mathcal{A}$ as the honest execution of the protocol. Then upon receiving the list of ciphertexts $c'_{\times,P_i(\boldsymbol{x})}$'s for the encryption switching procedure, $\mathcal{S}$ picks $c'_{+,P_i(\boldsymbol{x})} \leftarrow \Pi_+.\mathsf{Enc}(0)$ for all i and sends them to $\mathcal{A}$. When $\mathcal{A}$ calls $\mathcal{F}_{\mathsf{zk}}^{\mathsf{TwinCtx}}$ for these $c'_{\times,P_i(\boldsymbol{x})}$'s and $c'_{+,P_i(\boldsymbol{x})}$'s, $\mathcal{S}$ replies accept on behalf of $\mathcal{F}_{\mathsf{zk}}^{\mathsf{TwinCtx}}$ as the honest execution of the protocol. Finally, the simulator $\mathcal{S}$ outputs what $\mathcal{A}$ outputs to conclude the simulation.

We claim that the simulated view derived from simulation strategy above is computationally indistinguishable from the view of $\mathcal{A}$. Note that the difference between these two views is the distributions of ciphertext list of $c'_{+,P_i(\boldsymbol{x})}$'s. Let us define the following games to prove this claim.

Game 0**.** Suppose that $\mathcal{S}$ knows the private key $\mathsf{sk}_\times$. All $c_{+,P_i(\boldsymbol{x})}$'s in this game are computed by $\mathcal{S}$ as in a real execution of the generic construction. This game is the transcript of the real execution in the hybrid model.

Game i **(for** $i = 1, \ldots, k-1$**).** Suppose that $\mathcal{S}$ knows the private key $\mathsf{sk}_\times$. In this game, the first i ciphertexts are replaced by the ciphertexts computed as in the simulation, *i.e.*, for $j = 1, \ldots, i$, $c_{+,P_j(\boldsymbol{x})}$ is a random ciphertext that encrypts 0. The last $k-i$ ciphertexts $c_{+,P_i(\boldsymbol{x})}$'s are computed the same as the real execution.

Game k**.** This game is the simulation. All ciphertexts $c_{+,P_j(\boldsymbol{x})}$'s are random ciphertexts encrypting 0.

If there exists a distinguisher $\mathcal{D}$ that can distinguish **Game** 0 and **Game** k, there should exist an index $i \in \{0, \ldots, k-1\}$, such that it is possible to distinguish **Game** i and **Game** $i+1$. The difference between **Game** i and **Game** $i+1$ is whether the ciphertext $c_{+,P_{i+1}(\boldsymbol{x})}$ encrypts the value as in the real execution or 0. However, since Π_+ is IND-CPA secure, **Game** i and **Game** $i+1$ for all $i \in \{0, \ldots, k-1\}$ are computationally indistinguishable. Thus, **Game** 0 and **Game** k are computationally indistinguishable. It then follows that the generic construction in the hybrid model with the ideal functionalities $\mathcal{F}_{\mathsf{zk}}^{\mathsf{TwinCtx}}$, $\mathcal{F}_{\mathsf{zk}}^{\mathsf{EncValue}}$, and $\mathcal{F}_{\mathsf{zk}}^{\mathsf{sk}_+}$ achieves procedure obliviousness. □