zkMarket: Privacy-preserving Fair Data Trade System on Blockchain

Seungwoo Kim[1], Semin Han[2], Seongho Park[2], Kyeongtae Lee[2], Jihye Kim[1,3], and Hyunok Oh[2,3]

[1] Kookmin University, Seoul, South Korea
donny11489@gmail.com, jihyek@kookmin.ac.kr
[2] Hanyang University, Seoul, South Korea
{seminhan, seonghopark, rsias9049, hoh}@hanyang.ac.kr
[3] Zkrypto, Seoul, South Korea

Abstract. In this paper, we introduce zkMarket, a privacy-preserving fair trade system on the blockchain. zkMarket addresses the challenges of transaction privacy and computational efficiency. To ensure transaction privacy, zkMarket is built upon an anonymous transfer protocol. By combining encryption with zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARK), both the seller and the buyer are enabled to trade fairly. Furthermore, by encrypting the decryption key, we make the data registration process more concise and improve the seller's proving time by leveraging commit-and-prove SNARK (CP-SNARK) and our novel pseudorandom generator, the matrix-formed PRG (MatPRG).
Our evaluation demonstrates that zkMarket significantly reduces the computational overhead associated with traditional blockchain solutions while maintaining robust security and privacy. The seller can register 1MB of data in 3.2 seconds, while the buyer can generate the trade transaction in 0.2 seconds, and the seller can finalize the trade in 0.4 seconds.

1 Introduction

In the realm of digital data trading, a fundamental principle is ensuring fairness, where the seller receives the payment only if the data is delivered, and the buyer receives the data only if they pay the correct amount. Traditionally, a trusted third party (TTP) was considered essential for designing a fair trade [23]. Under this model, various approaches that depend on a TTP have been proposed [12, 19, 29]. However, with the emergence of blockchain technology, the advantages of immutability and transparency in blockchain along with a smart contract have garnered significant attention [22, 30]. A line of works has proposed blockchain-based fair trade protocols that eliminate the need for a TTP [9, 10, 13, 25–27, 31].

A major challenge in implementing blockchain-based data trading is balancing transparency with fairness. While transactions are publicly uploaded due to the transparency of the blockchain, this can allow individuals who have not paid to gain access to the data, potentially compromising fairness. Recent studies in data trading protocols on blockchains [2, 8, 20, 21] aim to construct fair trade protocol through a hashed time lock contract. Intuitively, the seller first encrypts the data and then trades the decryption key. The seller proves that the key is indeed the correct decryption key for the data using a zero-knowledge proof (ZKP) while the data remains undisclosed. Once the proof is verified, the key is revealed to solve the hashed time lock contract (HTLC), and their payment is concurrently sent to the seller.

The recent work SmartZKCP [21] highlights that this approach burdens the seller with extensive computational costs, potentially exposing the seller to Denial of Service (DoS) attacks when a malicious buyer repeatedly requests data. In traditional HTLC frameworks, to resolve the puzzle and receive a payment, the seller must disclose the decryption key. Since the key is not reusable, the seller should encrypt the data for every trade and generate the ZKP. SmartZKCP addresses this issue by re-encrypting the encrypted data using a pre-exchanged key. This method ensures that even if the original decryption key is revealed, only the buyer can decrypt the re-encrypted ciphertext and access the ciphertext of the original data. However, the seller still needs to re-encrypt the data for each trade. While SmartZKCP eliminates the necessity of iterative proving, it still requires the seller to compute encryption within the circuit, which scales with the data size. According to the experimental figure in [21, Section 6], the seller takes 20 seconds of proving time for 5KB data. It still presents challenges for real-world scenarios, where the data size is large (e.g., a short web novel is usually 20$\sim$40KB[4]).

Furthermore, the privacy issues introduced by blockchain's inherent transparency cannot be overlooked. Every transaction recorded on the blockchain is public, potentially exposing sensitive financial and behavioral patterns that could be exploited for advertising, marketing, or fraudulent activities. To address these concerns, a privacy-preserving data exchange protocol has been proposed [4]. With the circuit randomization technique [17], transaction details are effectively shielded from the smart contract and external parties. However, the runtime for the seller remains extremely high (about 28 seconds for 15 bytes), posing a significant challenge for real-world applications.

To address these issues, we introduce zkMarket, a privacy-preserving fair digital data trading system on blockchain. zkMarket ensures twofold fairness: 1) *Seller fairness*: A buyer cannot obtain any knowledge of data before they pay a fair price, and 2) *Buyer fairness*: A seller cannot receive the payment before they deliver the data to the buyer. The seller encrypts both the data and the decryption key. Subsequently, only once at the initial registration phase, the seller publishes the hash values of respective encrypted data and the decryption key on the blockchain along with a zk-SNARK proof to demonstrate that: 1) the hash values are indeed output of hashing the ciphertext and the decryption key, and 2) the ciphertext is the encryption of exactly what the buyer requests to purchase. Since only the buyer who pays can decrypt the ciphertext of the decryption key, seller fairness is guaranteed. Buyer fairness is also ensured, as the seller publishes the hash output of the ciphertext and the decryption key along with the proof. Specifically, the seller cannot deceive the buyer with misrepresented data. We also emphasize that the registration is required only once, it is concise and resilient to DoS attacks.

To maximize feasibility, we propose a registration algorithm for zkMarket, which is efficient even for large-sized data, making it compatible with real-world applications. The foundation of the zkMarket registration algorithm is adopting commit-and-prove SNARK (CP-SNARK), and our novel technique, matrix-formed pseudorandom gen-

[4] For the case of a long novel, the average size of Kindle book ranges from 2.7MB to 4.5MB [28]

erator (MatPRG). As mentioned earlier, the registration phase requires the seller to requires the seller to encode both the encryption and the hash computation, which incurs significant computational overhead, proportional to the data size. By leveraging CP-SNARK, which allows a commitment to the witness to be provided as input, the seller can avoid proving hash computation within the zk-SNARK circuit. Briefly, the seller computes the hash of the ciphertext using the Pedersen hash, and passes it as input to the zk-SNARK, allowing the seller to avoid encoding hash computation in the circuit. The seller can reduce the proving time for encryption by employing our new approach, MatPRG. MatPRG constructs a key matrix consisting of the decryption key and a seed. MatPRG multiplies it with a randomly chosen matrix, with the resulting matrix elements used as pseudorandom values. Proving encryption requires encoding a pseudorandom function in the circuit, which might involve encoding hash computations within the circuit when employing counter (CTR) mode [20]. By utilizing MatPRG as a pseudorandom function of symmetric key encryption, encoding hash computations can be replaced with a single matrix multiplication. Finally, the seller can reduce the cost of proving encryption, even for large-sized data. Our evaluation shows that the proving time for the registration of zkMarket is 0.57 seconds for 32KB of data, whereas SmartZKCP takes 20 seconds for a much smaller data size of 5 KB. We also observe that the proving time for 32MB data (which could be the size of a novel) is approximately 130 seconds. We argue that zkMarket maintains practicality even for much larger data as the registration is only required once.

Moreover, zkMarket offers anonymous trading, ensuring that transaction details are valid. This is achieved using encrypted accounts and zk-SNARK, which validate transaction integrity without revealing sensitive information. The system ensures that details such as the identities of the buyer and seller, the data's nature, and the payment amount remain protected, while the transaction's validity is publicly verifiable. For payment privacy, zkMarket integrates the existing anonymous transfer technique [18] to shield financial details.

1.1 Our contributions

As a result, zkMarket has been designed as a privacy-preserving and fair digital data trading platform where buyers and sellers can transact equitably, and no one can learn any transaction information, including the identities of the traders, and data details. Our contributions are summarized as follows:

- We propose zkMarket, which is a blockchain-based digital data trade platform providing fairness through combining encryption and zk-SNARK. zkMarket also enables the participants to trade anonymously by employing an anonymous transfer protocol.
- zkMarket supports one-time registration by encrypting the decryption key as well as the data. As a result, zkMarket is robust against DoS attacks.
- We also significantly reduce the proving cost of the seller incurred during the data registration by employing commit-and-prove SNARK (CP-SNARK) and devising a novel primitive MatPRG.
- We fully implement zkMarket and empirically evaluate the practicality of zkMarket. For instance, proving time for registering 32KB data takes approximately 0.19

seconds, and only 3.2 seconds is taken for 1MB data. We stress that these are practical figures since registration is required only once in the initial phase. Moreover, proving time for trade request and acceptance takes 0.2 seconds and 0.38 seconds respectively, regardless of the data size.

1.2 Related work

The Zero-Knowledge Contingent Payment (ZKCP) [2] protocol, leverages zero-knowledge succinct non-interactive argument of knowledge (zk-SNARK) and a hash-locked transaction to facilitate fair exchanges on the Bitcoin network. ZKCP achieves fairness by applying zk-SNARK to verify that encrypted data satisfies conditions defined by the buyer, without revealing the data itself. With this method, the seller has to create a verifiable commitment to their data and generate proofs that confirm the data satisfies the buyer's specified requirements. This proof guarantees payment only if the data aligns with the buyer's expectations. While the first implementation of ZKCP is instantiated with Pinocchio [24] zk-SNARK which needs a trusted setup, ZKCPlus [20] extends ZKCP by eliminating the trusted setup and improving the performance of sellers. They replace trusted setup with public setup and lessen the proving overhead by adopting circuit-friendly block cipher in a data-parallel encryption mode and devising a new commit-and-prove non-interactive zero-knowledge (CP-NIZK) argument of knowledge. SmartZKCP [21] identifies that the off-chain verification in ZKCP and ZKCPlus can cause a reputation attack, where a malicious third party could damage an honest seller's reputation by falsely claiming that the seller delivered incorrect goods or invalid proofs. SmartZKCP also identifies vulnerabilities in ZKCP, such as a DoS attack and an eavesdropper's attack. SmartZKCP proposes an advanced ZKCP protocol where the eavesdropper's attack is mitigated through double encryption while the DoS attack is prevented by locking the buyer's fee within the smart contract.

2 Preliminaries

We introduce the notations and (informal) definitions of cryptographic primitives used throughout this paper. We denote random sampling of an element by $\leftarrow\$$. For example, $x \leftarrow\$ \mathbb{F}$ denotes the random sampling x from finite field $\mathbb{F}$. A hash function denoted as CRH is the collision-resistant hash function.

2.1 Encryption schemes

We use standard definitions of a symmetric-key encryption scheme $\mathsf{SE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ and public-key encryption scheme $\mathsf{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$. Both encryption schemes ensure ciphertext indistinguishability under chosen-plaintext attack (IND-CPA) security and key indistinguishability under chosen-plaintext attack (IK-CPA [5]) security. Formal definitions are described in Appendix A.1 and Appendix A.2 respectively.

2.2 Commitments

We use a standard commitment scheme $\mathsf{Com} = (\mathsf{Setup}, \mathsf{Com})$ that allows one to commit a value. $\mathsf{Setup}(1^\lambda)$ outputs a commitment key ck taking the security parameter as input, and $\mathsf{Com}(\mathsf{ck}, m; o)$ returns the commitment c to message m over the opening randomness o with ck. A commitment scheme should ensure hiding where the committed value does not reveal any information about the value, and binding where the commitment is only opened to the original committed value.

2.3 Merkle Tree

Merkle Tree is a data structure where a party can commit to some value succinctly and further prove a membership of some leaf value. Briefly, each leaf is computed by (collision-resistant) hashing specific value (e.g., a commitment to some value) and its parent node is computed by hashing its children nodes, and the whole tree is constructed by working iteratively until reaching the root. rt denotes the root of the tree and the path for proving membership of a specific node (node) is denoted as $\mathsf{Path}_{\mathsf{node}}$. The algorithm consists of as following:

- $\mathsf{ComputePath}(\mathsf{node}) \rightarrow \mathsf{Path}$: takes a leaf node node as input and returns a corresponding authentication path Path to the root rt.
- $\mathsf{MemVerify}(\mathsf{rt}, \mathsf{node}, \mathsf{Path}_{\mathsf{node}}) \rightarrow 0/1$: takes the root rt, a leaf node node, and its corresponding membership proof $\mathsf{Path}_{\mathsf{node}}$ and outputs 1 if rt matches the hash value computed from node along with $\mathsf{Path}_{\mathsf{node}}$, 0 otherwise.
- $\mathsf{TreeUpdate}(\mathsf{node}_{\mathsf{new}}) \rightarrow \mathsf{rt}_{\mathsf{new}}$: returns new root value $\mathsf{rt}_{\mathsf{new}}$ for the updated Merkle Tree on newly added value new.

2.4 SNARK

The definition of a SNARK for a relation R is composed of a tuple of algorithms $\Pi_{\mathsf{snark}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ working as follows (A formal definition is described in Appendix A.3):

- $\mathsf{Setup}(1^\lambda, R) \rightarrow \mathsf{crs}$: takes a security parameter 1^λ and a relation R as inputs, and returns a common reference string crs.
- $\mathsf{Prove}(\mathsf{crs}, \mathbf{x}, \mathbf{w}) \rightarrow \pi$: outputs a proof π on inputs crs, a statement $\mathbf{x}$, and a witness $\mathbf{w}$ such that $R(\mathbf{x}; \mathbf{w})$.
- $\mathsf{Verify}(\mathsf{crs}, \mathbf{x}, \pi) \rightarrow 0/1$: inputs crs, $\mathbf{x}$ and π and outputs 1 if π is accepted, 0 otherwise.

2.5 Anonymous transfer protocol

To trade digital content on blockchain, a fee transaction occurs inevitably between the seller and the buyer. On blockchains like Ethereum, the transaction is opened to any party. It implies that anyone can observe transaction details, including payments. To ensure better privacy (referred to here as trade anonymity), the transaction must be hidden from unrelated parties. To provide user privacy on a public blockchain, we apply the anonymous transfer protocol such as Azeroth [18], zeroCash [6], and blockMaze [16]. In this paper, we employ Azeroth due to its advantages in the efficiency of anonymous transfer and gas consumption.

Revisit Azeroth. Azeroth consists of two types of accounts: an externally owned account (EOA), which is visible publicly, and an encrypted account (ENA). The sct values represent encrypted account balances, which are mapped to the ENA in the smart contract as $\mathtt{addr}$. Both accounts work on the blockchain, with their encrypted balances to ensure privacy. Also, When a new commitment ($\mathsf{cm}_{\mathsf{Azeroth}}$) is added to the Azeroth's Merkle Tree, it signifies the addition of a new transaction or balance update. When a user wants to send funds to a specific recipient, they update Azeroth's Merkle Tree with a commitment(cm) and encrypt the recipient's information alongside the transaction. This setup enables only the intended recipient, who can decrypt the ciphertext included in the transaction, to claim ownership of the uploaded cm within the Merkle Tree. Furthermore, as the recipient's information is encrypted in the transaction, third parties are unable to identify the destination of the funds, ensuring confidentiality in fund transfers.

3 zkMarket

In this section, we present the construction of zkMarket along with the three main stages of the protocol: Data registration, Trade generation, and Trade acceptance.

3.1 Overview

Before we delve into zkMarket, we outline its construction and the properties to be considered. In zkMarket, a seller registers the data they want to sell on the blockchain, and a buyer purchases it by paying an appropriate price. However, a seller might try to receive payment without delivering the correct contents; that is, the seller could send different data that is not what the buyer requests. Conversely, a buyer might attempt to obtain the data without payment. Consequently, we consider the following security properties to prevent such *malicious behaviors* of the seller and the buyer.

- **Fairness**: Fairness can be divided into two aspects, seller fairness and buyer fairness. The former states that any buyer cannot obtain the (partial or whole) data before they fulfill the payment, and the latter ensures that any seller cannot receive the payment without delivering the data that the buyer requests to purchase.
- **Trade Anonymity**: No party can gain knowledge of transaction details, including which data is traded, who buys or sells, and the trade amount.

zkMarket is designed into three main phases to satisfy the aforementioned security properties: 1) Register phase: the seller registers their data on the blockchain market, 2) Trade generation phase: the buyer requests to purchase the data, and 3) Trade acceptance phase: the seller approves the purchase request sent by the buyer.

In the registration phase, a seller encrypts the data to prevent malicious behavior by the buyer. Since only the ciphertext of the data is accessible, no one can obtain (or infer) the raw data without payment. Thus, zkMarket satisfies seller fairness through the encryption of the data. To simultaneously provide buyer fairness, the seller is compelled not to change the data after receiving payment. Therefore, the seller additionally submits the hash values of the decryption key and the encrypted data, along with a zk-SNARK proof. This proof verifies that the ciphertext is an accurate encryption of the initially registered data and that the hash is correctly derived from the decryption key. It ensures that a malicious seller cannot deceive buyers by misrepresenting the data, as any changes would fail the verification. Furthermore, the seller encrypts the decryption key with buyer's public key so as to prevent that no one but only the buyer can take the decryption key. The seller additionally generates the proof proving that the seller encrypts the decryption well. This work is required only once at the registration phase during whole protocol, it is resistant to denial-of-service (DoS) attacks. However, the seller must encode hashing and encryption operations within the circuit, which results in significant computational overheads, undermining practicality as data size increases. To mitigate overhead in the registration phase, we propose two approaches. First, we efficiently prove hash computations using commit-proving SNARK (CP-SNARK). Second, we introduce MatPRG, a matrix-formed pseudorandom generator, to improve the efficiency of the encryption proof.

During the trade generation phase, the buyer submits a trade request and demonstrates their ability to pay using zk-SNARK. The buyer locks the fees in a smart contract and updates the blockchain with the necessary transaction details. In the transaction details uploaded by the buyer, information about the content being purchased and the seller is encrypted, preventing third parties from deducing any specifics from the transaction itself. Since the buyer must prove that the amount they are paying matches the price (of the content), seller fairness is still guaranteed in zkMarket. From the buyer's transaction, information such as the data being requested to trade, the price of the data, and the remaining balance in the buyer's account remains hidden.

In the trade acceptance phase, the seller approves the trade request and sends the decryption key to the buyer via the blockchain. However, since the decryption key is transmitted through the blockchain where transactions are publicly recorded, a malicious participant could attempt to obtain the decryption key without payment. To prevent such a malicious event, the seller encrypts the decryption key with the buyer's public key before transmission. Furthermore, to ensure that the seller provides the correct decryption key, the seller proves that the encrypted decryption key can indeed decrypt the data that the buyer requests to trade. Consequently, only the buyer involved in the transaction can access the decryption key. In the seller's transaction for accepting a trade, no information about the decryption key is leaked. Only a valid buyer can decrypt the ciphertext of the key, ensuring that the data can be traded anonymously.

3.2 Analysis on Register phase

In the registration phase, a seller encrypts the data to prevent unauthorized parties who do not pay from accessing it. In other words, encryption provides zkMarket with seller fairness since no one can get the data from the ciphertext without payment. The seller also encrypts the decryption key making the seller register the data only once, which enhances the robustness against denial-of-service (DoS) attacks. Moreover, a seller must publish the hash output of the ciphertext and generate proof that the published hash output is correctly computed from the ciphertext. The hash value is registered to a valid data (goods) list if and only if the proof verification passes. By doing this, it is ensured that a malicious seller cannot deliver different data after receiving payment. Even though fairness is fulfilled by submitting the proof, it incurs expensive proving overhead since the hash computation is proven in the zk-SNARK circuit. In detail, a seller proves that: 1) the ciphertext ct is indeed the encryption of data, 2) h_{ct} is the hash output of ct, and 3) h_{k} is the hash output of the encryption key (of data) k and the seller's secret key $\mathsf{sk}^{\mathsf{seller}}$, that is, $\mathsf{h}_{\mathsf{k}} = \mathsf{CRH}(\mathsf{sk}^{\mathsf{seller}}||\mathsf{k})$. Namely, the encryption and hash computations are encoded within the circuit, which imposes heavy computation on the seller. We present two approaches to register efficiently. The first one is employing CP-SNARK, and the second one is our novel PRG, MatPRG.

Leveraging CP-SNARKs for efficient hash computation One of the significant overheads for the prover in the registration phase is checking h_{ct} within the zk-SNARK circuit. The hash computation is proportional to the data size. Namely, computing the hash function within the circuit demands a substantial amount of proving time, particularly for large-sized data. To reduce the proving time required for checking h_{ct}, we employ CP-SNARK introduced in [7]. A commit-and-prove SNARK (CP-SNARK) for a relation R consists of four algorithms Π_{cp} = (Setup, Prove, Verify, VerCommit) that works as follows:

- $\mathsf{Setup}(R) \rightarrow (\mathsf{ck}, \mathsf{ek}, \mathsf{vk}) \leftarrow$: takes a relation R as input and outputs a common reference string that includes a commitment key ck, an evaluation key ek, and a verification key vk.
- $\mathsf{Prove}(\mathsf{ek}, \mathbf{x}, \mathbf{w}) \rightarrow (\pi, c; o)$: takes an evaluation key ek, a statement $\mathbf{x}$, and a witness $\mathbf{w} := (u, \omega)$ such that the relation $\mathcal{R}$ holds as inputs, and outputs a proof π, a commitment c, and an opening o such that $\mathsf{VerCommit}(\mathsf{ck}, c, u, o) = 1$.
- $\mathsf{Verify}(\mathsf{vk}, \mathbf{x}, \pi, c) \rightarrow 0/1$: takes a verification key vk, a statement $\mathbf{x}$, a commitment c, and a proof π as inputs, and outputs 1 if $\mathbf{x}, c, \pi$ is within the relation $\mathcal{R}$, or 0 otherwise.
- $\mathsf{VerCommit}(\mathsf{ck}, c, u, o) \rightarrow 0/1$: takes a commitment key ck, a commitment c, a message u, and an opening o as inputs, and outputs 1 if the commitment opening is correct, or 0 otherwise.

Definition 1. *CP-SNARK satisfies completeness, succinctness, knowledge soundness, zero-knowledge, and binding.*

CP-SNARK allows a commitment to the witness to be provided as input along with the proof. By hashing the ciphertext using a Pedersen hash, it can be treated as a commitment to the ciphertext. Consequently, proving h_{ct} within the circuit is smoothly replaced by committing to ct instead. Finally, a seller can move the hash computation outside of the circuit, thus reducing the proving overhead.

MatPRG: Matrix-formed PRG Recall that a seller has to prove that the ciphertext of data is encrypted correctly in the zk-SNARK circuit during the registration phase. To prove encryption efficiently, we can consider a block cipher based on PRF using SNARK-friendly hash function such as MiMC7 [3] or Poseidon [14], i.e., $\mathrm{ct} = \mathrm{data} + \mathsf{PRF}_{\mathsf{k}}(r)$. However, it is still a burden for the seller, particularly when the size of the data is large (e.g., an image file). In particular, if the data is large, a seller must encode the PRF within the circuit for each block.

To resolve such hindrance, we devise a novel PRG, MatPRG, the matrix-formed PRG. To clairfy, given a randomly chosen matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, MatPRG for input message $\mathbf{K_1} \in \{0,1\}^{xk}$ outputs pseudorandom matrix $\mathbf{R} \in \{0,1\}^{nk \log q}$. $\mathbf{R}$ is secure under *Linear System Model (LSM)*, where an adversary attempts an attack solely using a linear system algorithm, which allows one to perform only linear operations on given matrices. Under a linear system algorithm, an adversary can find the message $\mathbf{K_1}$ from the pseudorandom matrix $\mathbf{R}$ with only negligible probability. Further details for the LSM are deferred to Appendix E.2

Replacing the PRF in encryption with MatPRG can alleviate the proving overhead effectively. The key is expressed as matrix $\mathbf{K_1}$ of size is $x \times k$. Then, $\mathbf{K_2}$, a randomly sampled $(m-x) \times k$ matrix, is generated in the $\mathsf{MatRand}$ phase. With $\mathbf{K_1}$, and $\mathbf{K_2}$, the matrix $\mathbf{K}$ is defined as $\begin{bmatrix} \mathbf{K_1} \\ \mathbf{K_2} \end{bmatrix}$, and the pseudorandom matrix $\mathbf{R}$ is generated as $\mathbf{R} \leftarrow f_{\mathbf{A}}(\mathbf{K}) \in \{0,1\}^{nk \log q}$ by the MatPRG. The formal definition of MatPRG is as follows.

Definition 2. *Let $n, m, q \in \mathbb{N}$ such that $m > n$ and $m \approx n$. Let $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and $\mathbf{K} \in \{0,1\}^{m \times k}$. Then, we define a function $f_{\mathbf{A}} : \{0,1\}^{mk} \to \mathbb{Z}_q^{n \times k} \cong \{0,1\}^{nk \log q}$ as $\mathbf{K} \mapsto \mathbf{AK}$.*

Since $m \approx n$, the function $f_{\mathbf{A}}$ is not surjective. That is, for arbitrary $\mathbf{R} \in \mathbb{Z}_q^{n \times k}$, there may not exist $\mathbf{K} := f_{\mathbf{A}}^{-1}(\mathbf{R}) \in \{0,1\}^{m \times k}$. However, generally it is difficult to determine whether $\mathbf{K}$ exists or not because computing $\mathbf{K}$ from the given random matrix $\mathbf{A}$ and $\mathbf{R}$ is challenging.

Definition 3 **(MatPRG).** *Let $\mathbf{K_1} \in \{0,1\}^{x \times k}$ be a key. Let $\mathbf{A} \leftarrow\!\!\$\ \mathbb{Z}_q^{n \times m}$ be a randomly selected matrix. Then the MatPRG is defined by the following process:*

- $\mathbf{K}_2 \leftarrow \mathsf{MatRand}(1^\lambda)$*: Generate $\mathbf{K_2} \leftarrow\!\!\$\ \{0,1\}^{(m-x)k}$.*
- $\mathbf{R} \leftarrow \mathsf{MatPRG}(\mathbf{A}, \mathbf{K}_1, \mathbf{K}_2)$*: The verifier takes $\mathbf{K} := \begin{bmatrix} \mathbf{K_1} \\ \mathbf{K_2} \end{bmatrix} \in \{0,1\}^{mk}$ as input and computes the output $\mathbf{R} = f_{\mathbf{A}}(\mathbf{K}) \in \{0,1\}^{nk \log q}$.*

Theorem 1. *If $m = n + \delta$ and $\delta k > 128$, then the MatPRG is a pseudorandom generator and it is secure under the LSM.*

Given the matrix formed symmetric key $\mathbf{K}_1$, $\mathbf{K}$ is constructed by combining $\mathbf{K}_1$ and the seed $\mathbf{K}_2$, while $\mathbf{R}$ is the matrix of random values generated by MatPRG. With MatPRG, a seller can efficiently prove $\mathsf{ct}_i = \mathsf{data}_i + \mathsf{PRF}_{\mathsf{k}}(r)$. Since each element of $\mathbf{R}$ is used as $\mathsf{PRF}_{\mathsf{k}}(r)$, proving the encryption in the registration phase is reduced to proving that $\mathbf{K}$ is constructed by combining the key $\mathbf{K}_1$ and the seed $\mathbf{K}_2$, and that $\mathbf{R}$ is the matrix of random values generated by MatPRG. In other words, proving the PRF for each ciphertext is replaced to compute one matrix multiplication. The detailed relation for R_{ct} with MatPRG is available in Appendix D.1

3.3 Construction

Putting everything together, we present the construction of zkMarket. The algorithm with prefix SC. denotes the algorithms executed on the smart contract. The overview of zkMarket is illustrated in Figure 1 and the overall flow based on subsequent algorithms is available in Figure 3 in Appendix F.

Setup phase The setup algorithms (Algorithm 1) for zkMarket run both off-chain and on the smart contract. In the off-chain setup, Setup generates a common reference string (CRS) for zk-SNARK of three relations corresponding to the three main phases: R_{reg}, R_{gen}, and R_{acc}, and also generates the key pair $(\mathsf{pk}, \mathsf{sk})$ for public key encryption. It also invokes the setup of $\mathsf{Azeroth}$ for anonymous transfer and outputs an address addr and the symmetric key $\mathsf{k}^{\mathsf{ENA}}$ used for encrypting the account. Since the smart contract executes verification for zk-SNARK proofs, $\mathsf{SC.Setup}$, therefore, stores verification keys

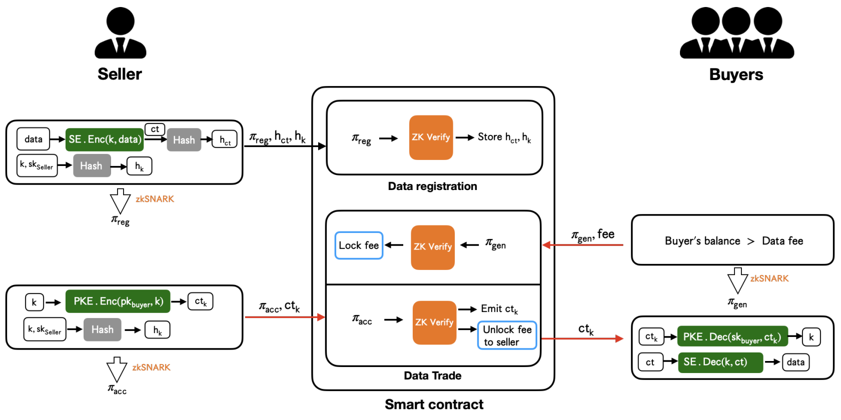

Fig. 1: The overview of zkMarket. Transactions depicted in red color represent the Azeroth transactions.

$(\mathsf{vk}_{\mathsf{reg}}, \mathsf{vk}_{\mathsf{gen}}, \mathsf{vk}_{\mathsf{acc}})$. Finally, a Merkle tree is initialized to facilitate anonymous transactions. We defer detailed descriptions of the relations of each algorithm to Appendix D due to the space limit.

Algorithm 1 Setup Algorithm

```
Off − chain
Setup($1^\lambda$) :
$\mathsf{crs}_1 \leftarrow \Pi_{\mathsf{cp}}.\mathsf{Setup}(1^\lambda, \mathcal{R}_{\mathsf{reg}})$
$\mathsf{crs}_2 \leftarrow \Pi_{\mathsf{snark}}.\mathsf{Setup}(1^\lambda, \mathcal{R}_{\mathsf{gen}})$
$\mathsf{crs}_3 \leftarrow \Pi_{\mathsf{snark}}.\mathsf{Setup}(1^\lambda, \mathcal{R}_{\mathsf{acc}})$
$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.Gen}(1^\lambda)$
$(\mathsf{addr}, \mathsf{k}^{\mathsf{ENA}}) \leftarrow \mathsf{Azeroth.Setup}(1^\lambda)$;
return $\mathsf{crs} := (\mathsf{pk}, \mathsf{sk}, \mathsf{addr}, \mathsf{k}^{\mathsf{ENA}}, \mathsf{crs}_1, \mathsf{crs}_2, \mathsf{crs}_3)$

Smart Contract
SC.Setup(pp) :
Store $\mathsf{vk}_{\mathsf{reg}}, \mathsf{vk}_{\mathsf{gen}}, \mathsf{vk}_{\mathsf{acc}}$;
Initialize a Merkle Tree MT;
```

Registration phase As described in Section 3.2, the one-time registration phase can provide seller fairness and buyer fairness by encrypting the data and the decryption key and publishing the corresponding hash values. Also, the proving overhead of the seller is mitigated by employing CP-SNARK and MatPRG. By leveraging CP-SNARK, the seller no longer needs to encode hash computation for h_{ct} within the circuit, and this approach allows the commitment to h_{ct} to be output during proof generation. Moreover, with MatPRG, the seller can prove the correctness of encryption in a batch manner.
Off-chain: In RegisterData , the seller encrypts the data (data) with symmetric key encryption and publishes hash outputs of ciphertext h_{ct} and the corresponding key h_{k}. Then, the seller generates a proof π_{reg} to prove that: 1) the data is correctly encrypted, and 2) the hashed values h_{k} and h_{ct} are correctly computed from the ciphertext and the

key corresponding to the data that the seller is registering. Seller-fairness is guaranteed by the encryption of data, since anyone on the blockchain, including the buyer, cannot obtain the data before paying. Buyer-fairness can be preserved by the hash outputs of the key and ciphertext; a malicious seller cannot replace the data with different or incorrect data after registration, since the hash outputs of the ciphertext and the key are proven under the zk-SNARK. Note that invalid or altered data cannot pass the verification of zk-SNARK.
Smart contract: The smart contract for data registration allows a seller to register data on the blockchain if and only if the proof published by the seller is valid. It verifies the proof π_{reg} along with $\mathsf{h_k}$, $\mathsf{h_{ct}}$, and adds those hash outputs to $\mathsf{List_{data}}$.

Algorithm 2 RegisterData Algorithm

```
Off − chain
RegisterData $(1^\lambda, \mathsf{crs}, \mathsf{data}, \mathsf{sk}^{\mathsf{seller}}, \mathbf{A} \in \mathbb{Z}_q^{n\times m})$ :
$\mathbf{K}_1 \in \{0,1\}^{x\times k} \leftarrow \mathsf{SE.Gen}(1^\lambda)$
$(\mathbf{K}_2, \mathsf{ct}) \leftarrow \mathsf{SE.Enc}(\mathbf{K}_1, \mathbf{A}, \mathsf{data})$
$\mathbf{K} := \begin{bmatrix}\mathbf{K}_1\\ \mathbf{K}_2\end{bmatrix} \in \{0,1\}^{m\times k}$
$\mathsf{h_k} \leftarrow \mathsf{CRH}(\mathsf{sk}^{\mathsf{seller}}||\mathsf{k})$
$\gamma \leftarrow\!\$\ \mathbb{Z}_q^{k\times 1}$
$\mathbf{x} := (\mathbf{A}, \mathsf{h_k})$
$\mathbf{w} := (\mathsf{ct}, \mathsf{data}, \mathbf{K}, \mathbf{R}, \gamma, \mathsf{sk}^{\mathsf{seller}})$
$(\pi_{reg}, \mathsf{h_{ct}}) \leftarrow \Pi_{\mathsf{cp}}.\mathsf{Prove}(\mathsf{crs}_1, \mathbf{x}; \mathbf{w})$
return $\mathsf{tx}_{reg} = (\mathsf{h_k}, \mathsf{h_{ct}}, \pi_{reg})$

SE.Enc$(\mathbf{K}_1, \mathbf{A}, \mathsf{data})$
$\mathbf{K}_2 \leftarrow \mathsf{MatRand}(1^\lambda)$
$\mathbf{K} := \begin{bmatrix}\mathbf{K}_1\\ \mathbf{K}_2\end{bmatrix} \in \{0,1\}^{m\times k}$
$\mathbf{R} \in \mathbb{Z}_q^{n\times k} \leftarrow \mathsf{MatPRG}(\mathbf{A}, \mathbf{K}_1, \mathbf{K}_2)$
for $i$ in $\{0, ..., n\}$ do
    for $j$ in $\{0, ..., k\}$ do
        $\mathsf{ct}[i+j] \leftarrow \mathsf{data}[i+j] + \mathbf{R}[i][j]$
return $(\mathbf{K}_2, \mathsf{ct})$

Smart Contract
SC.RegisterData $(\mathsf{tx_{reg}})$ :
parse $\mathsf{tx_{reg}} = (\mathbf{x}, \pi_{\mathsf{reg}})$;
parse $\mathbf{x} := \mathsf{h_k}, \mathsf{h_{ct}}$;
assert $\Pi_{\mathsf{cp}}.\mathsf{Verify}(\mathsf{vk_{reg}}, \mathbf{x}, \pi_{\mathsf{reg}})$;
$\mathsf{List_{data}} \leftarrow \mathsf{List_{data}} \cup \{\mathsf{h_k}, \mathsf{h_{ct}}\}$;
```

Trade generation phase A buyer can request an order for data registered on the market during the trade generation phase. Through GenerateTrade, a buyer commits to its payment for the purchase. At the same time, the buyer encrypts an order including the payment details. Then, it sends the transaction with proof proving that the payment is correctly committed and the ciphertext is indeed the encryption of the order made by the buyer. The proof additionally demonstrates that the balance of the buyer's account is sufficient to make the payment.
Off-chain: The buyer expresses its intention to purchase using the GenerateTrade (Algorithm 3), which comprises three primary steps.

First, the buyer commits to its payment. Specifically, the buyer commits to its payment fee using the seller's public key $\mathsf{pk}^{\mathsf{seller}}$, the hash of the symmetric key $\mathsf{h_k}$, the buyer's public key $\mathsf{pk}^{\mathsf{buyer}}$, and the randomness r. The commitment cm is then published on the blockchain. By committing to the payment, even a malicious buyer cannot deny the purchase or decrease the payment amount below the agreed price.

Secondly, the buyer encrypts an order, $\mathsf{order} := (r, \mathsf{fee}, \mathsf{h_k}, \mathsf{pk}^{\mathsf{buyer}})$. Encrypting order prevents cheating by the seller, where a malicious seller might take fee before delivering data.

Finally, to transfer anonymously, the buyer creates a new encrypted account state $\mathsf{sct}_{\mathsf{new}}$ to vindicate its payment capacity. The previous encrypted state $\mathsf{sct}_{\mathsf{old}}$ represents the buyer's existing balance, while $\mathsf{sct}_{\mathsf{new}}$ indicates the remaining balance after deducting the payment.

The proof π_{gen} proves that: 1) the commitment cm is indeed a commitment to $\mathsf{pk}^{\mathsf{seller}}$, fee, $\mathsf{h_k}$, $\mathsf{pk}^{\mathsf{buyer}}$ and r; 2) $\mathsf{ct}_{\mathsf{order}}$ is the ciphertext resulting from encrypting order under $\mathsf{pk}^{\mathsf{seller}}$; and 3) the value of fee is equal to difference between the decrypted values of $\mathsf{sct}_{\mathsf{old}}$ and $\mathsf{sct}_{\mathsf{new}}$.

Smart contract: The SC.GenerateTrade (Algorithm 3) handles the buyer's requests to purchase the data registered on the blockchain. It first verifies that $\mathsf{sct}_{\mathsf{old}}$ used in proof generation matches the value registered on the blockchain, and then it validates the proof π_{gen}. Once the proof verification is passed, it updates $\mathsf{sct}_{\mathsf{old}}$ to $\mathsf{sct}_{\mathsf{new}}$ and updates cm to the Merkle tree MT. During this process, the buyer's balance is updated to reflect the deduction of the purchase cost. Finally, it emits an event[5] with $\mathsf{ct}_{\mathsf{order}}$ to enable the seller to process the order.

Algorithm 3 GenerateTrade Algorithm

```
Off−chain
$\mathsf{GenerateTrade}(1^\lambda, \mathsf{crs}, \mathsf{fee}, \mathsf{pk}^{\mathsf{seller}}, \mathsf{pk}^{\mathsf{buyer}}, \mathsf{addr}^{\mathsf{buyer}}, \mathsf{h_k}, \mathsf{k_{ENA}})$ :
$r \leftarrow\!\$\ \mathbb{F}$
$\mathsf{cm} \leftarrow \mathsf{Com}(\mathsf{pk}^{\mathsf{seller}} || \mathsf{fee} || \mathsf{h_k} || \mathsf{pk}^{\mathsf{buyer}}; r)$
$\mathsf{order} := (r, \mathsf{fee}, \mathsf{h_k}, \mathsf{pk}^{\mathsf{buyer}})$
$\mathsf{ct_{order}} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}^{\mathsf{seller}}, \mathsf{order})$
$\mathsf{sct_{old}} \leftarrow \mathsf{ENA}[\mathsf{addr}^{\mathsf{buyer}}]$
$\mathsf{bal_{old}} \leftarrow \mathsf{SE.Dec}(\mathsf{k}^{\mathsf{ENA}}, \mathsf{sct_{old}})$
$\mathsf{bal_{new}} \leftarrow \mathsf{bal_{new}} - \mathsf{fee}$
$\mathsf{sct_{new}} \leftarrow \mathsf{SE.Enc}(\mathsf{k}^{\mathsf{ENA}}, \mathsf{bal_{new}})$
$\mathbf{x} := (\mathsf{cm}, \mathsf{ct_{order}}, \mathsf{sct_{old}}, \mathsf{sct_{new}})$
$\mathbf{w} := (r, \mathsf{h_k}, \mathsf{pk}^{\mathsf{seller}}, \mathsf{pk}^{\mathsf{buyer}}, \mathsf{k}^{\mathsf{ENA}}, \mathsf{fee})$
$\pi_{\mathsf{gen}} \leftarrow \Pi_{\mathsf{snark}}.\mathsf{Prove}(\mathsf{crs}_2, \mathbf{x}; \mathbf{w})$
return $\mathsf{tx_{gen}} = (\mathbf{x}, \pi_{\mathsf{gen}})$

Smart Contract
$\mathsf{SC.GenerateTrade}(\mathsf{tx_{gen}})$ :
parse $\mathsf{tx_{gen}} = (\mathbf{x}, \pi_{\mathsf{gen}})$;
parse $\mathbf{x} = (\mathsf{cm}, \mathsf{ct_{order}}, \mathsf{sct_{old}}, \mathsf{sct_{new}})$;
assert $\mathsf{ENA}[\mathsf{addr}^{\mathsf{buyer}}] == \mathsf{sct_{old}}$
assert $\Pi_{\mathsf{snark}}.\mathsf{Verify}(\mathsf{vk_{gen}}, \mathbf{x}, \pi_{\mathsf{gen}})$;
$\mathsf{ENA}[\mathsf{addr}^{\mathsf{buyer}}] \leftarrow \mathsf{sct_{new}}$
$\mathsf{rt_{new}} \leftarrow \mathsf{MT.TreeUpdate}(\mathsf{cm})$
$\mathsf{List_{rt}} \leftarrow \mathsf{List_{rt}} \cup \mathsf{rt_{new}}$
Emit Event $\mathsf{ct_{order}}$;
```

[5] Emit event is a mechanism that allows external applications to observe specific actions or state changes within a smart contract. These events are recorded on the blockchain.

Trade acceptance phase After the buyer sets up the trade through $\mathsf{GenerateTrade}$ and the proof is verified by the smart contract using $\mathsf{SC.GenerateTrade}$, the seller decides whether to accept the request. The seller first checks that the payment amount fee from the buyer is sufficient, i.e., the buyer has paid enough money to purchase the data. Then, to finalize the deal, the seller prepares the decryption key for the encrypted data which is to be delivered to the buyer, and claims the frozen fee from the buyer. The algorithms for the trade acceptance phase is depicted in Algorithm 4.
Off-chain: Through $\mathsf{AcceptTrade}$, the seller approves the buyer's request for purchase and finalizes the deal by providing the decryption key for the encrypted data ct. To accept, the seller decrypts the ciphertext of an order $\mathsf{ct}_{\mathsf{order}}$ using their secret key $\mathsf{sk}^{\mathsf{seller}}$. If the fee in order equals the price of the data, then the seller proceeds to send the decryption key k. However, the decryption key can be intercepted during the transmission. Therefore, the seller encrypts the key k as ct_{k} using the buyer's public key and generates a proof demonstrating that ct_{k} is indeed the encryption of the k that can decrypt the ct. We stress that this approach is advantageous as it does not require any trusted setting such as a secure channel. Subsequently, the seller computes the path Path of the commitment cm within the Merkle tree registered on the blockchain.

Additionally, the seller generates a nullifier nf using cm and $\mathsf{sk}^{\mathsf{seller}}$. If nf is not used, the seller could execute the $\mathsf{AcceptTrade}$ algorithm multiple times for the same purchase request, potentially receiving the fee multiple times. However, by generating nf using cm and the seller's secret key $\mathsf{sk}^{\mathsf{seller}}$, and checking that nf is not included in the $\mathsf{List}_{\mathsf{nf}}$ in the smart contract, the seller is prevented from executing the $\mathsf{AcceptTrade}$ algorithm more than once for the same request. This ensures that the seller cannot receive the fee more than once for the same transaction, preventing double-spending.

Next, the seller generates $\mathsf{o}_{\mathsf{Azeroth}}$ and $\mathsf{cm}_{\mathsf{Azeroth}}$ for claiming the fee in an anonymous transfer manner. Note that $\mathsf{o}_{\mathsf{Azeroth}}$ and $\mathsf{cm}_{\mathsf{Azeroth}}$ are required for anonymous transfer in Azeroth [18]. Briefly, using $\mathsf{cm}_{\mathsf{Azeroth}}$, the seller can anonymously validate their claim for fee, and $\mathsf{cm}_{\mathsf{Azeroth}}$ is used to update the Azeroth Merkle tree $\mathsf{MT}_{\mathsf{Azeroth}}$ after the proof π_{acc} is verified. Once the commitment $\mathsf{cm}_{\mathsf{Azeroth}}$ is updated in the Azeroth Merkle tree, the seller can retrieve the fee using the Azeroth protocol at any time in the future.[6].

Finally, the seller creates a commitment cm identical to the one generated by the buyer during the trade generation phase. Since the seller knows the messages of cm, they can prove that the same cm can be generated during the $\mathsf{AcceptTrade}$ step. Note that if cm is passed as a part of the statement for making the proof, anyone not involved in the trade can learn the identity of the buyer by linking the cm created by the buyer to the cm created by the seller. As a result, passing cm as part of the statement can reveal the identities of the trading participants. To prevent such linkability, we employ the Merkle tree to enable the buyer to verify the value and the membership of cm without including cm in the statement.

Consequently, π_{acc} proves that ct_{k} is genuinely the encryption of k which is used by the buyer to decrypt ct. Moreover, since we employ Azeroth as the anonymous transfer framework, the seller proves that the commitment $\mathsf{cm}_{\mathsf{Azeroth}}$ is computed correctly. Also,

[6] The $\mathsf{cm}_{\mathsf{Azeroth}}$ included in the Merkle tree can be transferred to the user's ENA using the zk-Transfer algorithm in Azeroth. Further details of the anonymous transfer are outside the scope of our interest. We refer to [18] for more detail.

as described previously, π_{acc} includes proving the Merkle tree membership for cm. Additionally, it ensures that the acceptance of the transaction is processed only once by verifying that nf is not in the $\mathsf{List_{nf}}$; i.e., by confirming that it is the initial transaction through a nullifier nf, it prevents the seller from accepting multiple fees from the same transaction.

Smart contract: SC.AcceptTrade allows the seller to accept purchase requests. First, it verifies the proof π_{acc} generated by the seller. Then, it checks whether nf is not in the $\mathsf{List_{nf}}$ and whether the root rt is in the $\mathsf{List_{rt}}$. Once these are completed, nf is append to the $\mathsf{List_{nf}}$, and the Azeroth Merkle tree is updated with $\mathsf{cm_{Azeroth}}$. Finally, it emits an event with $\mathsf{ct_k}$ to enable the buyer to retrieve the decryption key.

Algorithm 4 AcceptTrade Algorithm

```
Off − chain
AcceptTrade($1^\lambda$, crs, fee, $ct_{order}$, $pk^{seller}$, $sk^{seller}$, $pk^{buyer}$, $h_k$) :
order ← PKE.Dec($sk^{seller}$, $ct_{order}$)
parse order = ($r$, fee, $h_k$, $pk^{buyer}$)
$ct_k$ ← PKE.Enc($pk^{buyer}$, k)
cm ← Com($pk^{seller}$||fee||$h_k$||$pk^{buyer}$; $r$)
rt ← $List_{rt}$.TOP
Path ← MT.ComputePath(cm)
nf ← CRH(cm||$sk^{seller}$)
$o_{Azeroth}$ ←$ $\mathbb{F}$
$cm_{Azeroth}$ ← Com(fee||$addr^{seller}$ ; $o_{Azeroth}$)
x := (rt, nf, $cm_{Azeroth}$, $h_k$, $ct_k$, $pk^{seller}$, $addr^{seller}$)
w := (cm, Path, $sk^{seller}$, k, $pk^{buyer}$, $r$, fee, $o_{Azeroth}$)
$\pi_{acc}$ ← $\Pi_{snark}$.Prove($crs_3$, x; w)
return $tx_{acc}$ = (x, $\pi_{acc}$)

Smart Contract
SC.AcceptTrade($tx_{acc}$) :
parse $tx_{acc}$ = (x, $\pi_{acc}$);
parse x = (rt, nf, $cm_{Azeroth}$, $h_k$, $ct_k$, $pk^{seller}$, $addr^{seller}$);
assert nf $\notin$ $List_{nf}$
assert rt $\in$ $List_{rt}$
assert $\Pi_{snark}$.Verify($vk_{acc}$, x, $\pi_{acc}$);
$List_{nf}$ ← $List_{nf}$ ∪ {nf};
$MT_{Azeroth}$.TreeUpdate($cm_{Azeroth}$)
Emit Event $ct_k$;
```

4 Evaluation

4.1 Implementation

We implement zkMarket using the Arkworks library [11] in Rust and the smart contracts have deployed on the Ethereum test network blockchain using Hardhat [1]. We instantiate zk-SNARK with Groth16 [15] (for GenerateTrade and AcceptTrade described in Algorithm 3 and Algorithm 4), and cp-SNARK (for RegisterData described in Algorithm 2) from LegoSNARK [7] based on Groth16. Both of them work over a bilinear map. We use the BN254 for the curve instantiation. For the public key encryption, we employ the ElGamal encryption. We use the MiMC7 [3], SNARK-friendly hash function, and instantiate the Merkle tree based on MiMC7. All the benchmarks are evaluated using an Apple M1 Pro processor with 32GB of RAM.

4.2 Benchmarks for zkMarket

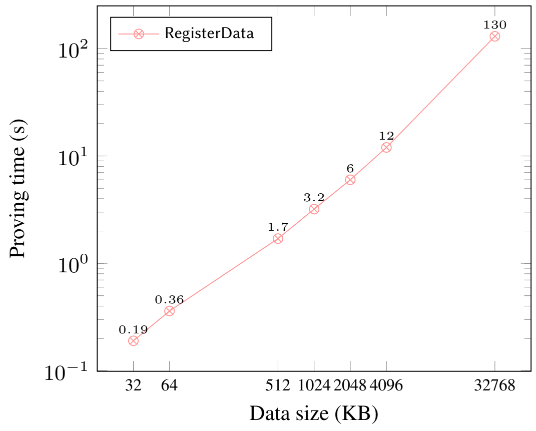

Fig. 2: Proving time of RegisterData

Performance analysis on data register As described above, the significant overhead in zkMarket emerges when the data is registered. Note that the performance of algorithms varies with the data size since the encryption and hash operations within the zk-SNARK circuit scale with the size of the data. The overhead for hash operations was resolved using CP-SNARK, while the overhead for encryption operations was addressed with MatPRG. As a result, proving 64KB of data takes 0.36 seconds, while 1024KB (1MB) requires 3.2 seconds of proving time. A 17-page PDF file of 2MB requires a proving time of 6 seconds, while a 36MB video file with 1080×1920 resolution (FHD) and a duration of 33 seconds results in a proving time of 130 seconds.

Performance evaluation of GenerateTrade and AcceptTrade Table 1 shows the performance evaluation of GenerateTrade and AcceptTrade with a Merkle tree of depth 32. Unlike RegisterData , GenerateTrade and AcceptTrade are executed each time a transaction occurs. However, GenerateTrade and AcceptTrade are not impacted by the size of the data, whereas RegisterData is. Recall that GenerateTrade proves the buyer's ability to pay, while AcceptTrade proves the correctness of the decryption key, both generating zk-SNARK proofs for a fixed-size fee and key. Consequently, GenerateTrade and AcceptTrade can run in constant time. The proving time and the verification time of GenerateTrade take around 200ms and 20ms respectively, and those of AcceptTrade take around 380ms and 0.02ms respectively.

Algorithm	Constraints	CRS size (MB)	Setup (s)	Prove (s)	Verify (s)
GenerateTrade	12,882	4.3	0.19	0.2	0.02
AcceptTrade	24,210	9.3	0.2	0.38	0.02

Table 1: Evaluation of GenerateTrade and AcceptTrade with 32 depth Merkle tree

Gas consumption of smart contract We also measure the gas consumption of smart contracts throughout whole phases in zkMarket. SC.RegisterData requires 285,131 gas, while SC.GenerateTrade and SC.AcceptTrade which include updating the Merkle tree, costs 1,996,915 and 1,378,750 respectively.

References

1. https://github.com/NomicFoundation/hardhat
2. Zero knowledge contingent payment. https://en.bitcoin.it/wiki/Zero_Knowledge_Contingent_Payment, accessed: 2024-06-17
3. Albrecht, M.R., Grassi, L., Rechberger, C., Roy, A., Tiessen, T.: Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In: ASIACRYPT. pp. 191–219 (2016)
4. Avizheh, S., Haffey, P., Safavi-Naini, R.: Privacy-preserving fairswap: Fairness and privacy interplay. Proceedings on Privacy Enhancing Technologies (2022)
5. Bellare, M., Boldyreva, A., Desai, A., Pointcheval, D.: Key-privacy in public-key encryption. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 566–582. Springer (2001)
6. Ben Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: 2014 IEEE Symposium on Security and Privacy. pp. 459–474 (2014). https://doi.org/10.1109/SP.2014.36
7. Campanelli, M., Fiore, D., Querol, A.: Legosnark: Modular design and composition of succinct zero-knowledge proofs. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 2075–2092 (2019)
8. Campanelli, M., Gennaro, R., Goldfeder, S., Nizzardo, L.: Zero-knowledge contingent payments revisited: Attacks and payments for services. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 229–243 (2017)
9. Chenli, C., Tang, W., Jung, T.: Fairtrade: Efficient atomic exchange-based fair exchange protocol for digital data trading. In: 2021 IEEE International Conference on Blockchain (Blockchain). pp. 38–46. IEEE (2021)
10. Chenli, C., Tang, W., Lee, H., Jung, T.: Fair 2 trade: Digital trading platform ensuring exchange and distribution fairness. IEEE Transactions on Dependable and Secure Computing (2024)
11. arkworks contributors: arkworks zksnark ecosystem (2022), https://arkworks.rs
12. Dai, W., Dai, C., Choo, K.K.R., Cui, C., Zou, D., Jin, H.: Sdte: A secure blockchain-based data trading ecosystem. IEEE Transactions on Information Forensics and Security **15**, 725–737 (2019)
13. Dziembowski, S., Eckey, L., Faust, S.: Fairswap: How to fairly exchange digital goods. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 967–984 (2018)
14. Grassi, L., Khovratovich, D., Rechberger, C., Roy, A., Schofnegger, M.: Poseidon: A new hash function for {Zero-Knowledge} proof systems. In: 30th USENIX Security Symposium (USENIX Security 21). pp. 519–535 (2021)
15. Groth, J.: On the size of pairing-based non-interactive arguments. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 305–326. Springer (2016)
16. Guan, Z., Wan, Z., Yang, Y., Zhou, Y., Huang, B.: Blockmaze: An efficient privacy-preserving account-model blockchain based on zk-snarks. Cryptology ePrint Archive, Paper 2019/1354 (2019), https://eprint.iacr.org/2019/1354, https://eprint.iacr.org/2019/1354
17. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: Securing hardware against probing attacks. In: Advances in Cryptology-CRYPTO 2003: 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003. Proceedings 23. pp. 463–481. Springer (2003)
18. Jeong, G., Lee, N., Kim, J., Oh, H.: Azeroth: Auditable zero-knowledge transactions in smart contracts. IEEE Access **11**, 56463–56480 (2023)

19. Jung, T., Li, X.Y., Huang, W., Qian, J., Chen, L., Han, J., Hou, J., Su, C.: Accounttrade: Accountable protocols for big data trading against dishonest consumers. In: IEEE INFOCOM 2017-IEEE Conference on Computer Communications. pp. 1–9. IEEE (2017)
20. Li, Y., Ye, C., Hu, Y., Morpheus, I., Guo, Y., Zhang, C., Zhang, Y., Sun, Z., Lu, Y., Wang, H.: Zkcplus: Optimized fair-exchange protocol supporting practical and flexible data exchange. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. pp. 3002–3021 (2021)
21. Liu, X., Zhang, J., Wang, Y., Yang, X., Yang, X.: Smartzkcp: Towards practical data exchange marketplace against active attacks. Cryptology ePrint Archive (2024)
22. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Decentralized Business Review p. 21260 (2008)
23. Pagnia, H., Gärtner, F.C., et al.: On the impossibility of fair exchange without a trusted third party. Tech. rep., Citeseer (1999)
24. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: Nearly practical verifiable computation. Communications of the ACM **59**(2), 103–112 (2016)
25. Sheng, D., Xiao, M., Liu, A., Zou, X., An, B., Zhang, S.: Cpchain: a copyright-preserving crowdsourcing data trading framework based on blockchain. In: 2020 29th international conference on computer communications and networks (ICCCN). pp. 1–9. IEEE (2020)
26. Su, G., Yang, W., Luo, Z., Zhang, Y., Bai, Z., Zhu, Y.: Bdtf: A blockchain-based data trading framework with trusted execution environment. In: 2020 16th International Conference on Mobility, Sensing and Networking (MSN). pp. 92–97. IEEE (2020)
27. Tas, E.N., Seres, I.A., Zhang, Y., Melczer, M., Kelkar, M., Bonneau, J., Nikolaenko, V.: Atomic and fair data exchange via blockchain. Cryptology ePrint Archive (2024)
28. The book buff: How Many Books Can a Kindle Hold? [8GB vs 32GB] (2023), `https://thebookbuff.com/how-many-books-can-a-kindle-hold/`
29. Wang, B., Li, B., Yuan, Y., Dai, C., Wu, Y., Zheng, W.: Cpdt: A copyright-preserving data trading scheme based on smart contracts and perceptual hashing. In: 2022 IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom). pp. 968–975. IEEE (2022)
30. Wood, G., et al.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper **151**(2014), 1–32 (2014)
31. Zhao, Y., Yu, Y., Li, Y., Han, G., Du, X.: Machine learning based privacy-preserving fair data trading in big data market. Information Sciences **478**, 449–460 (2019)

A Formal definitions

A.1 Symmetric-key encryption

We use a symmetric-key encryption scheme $\mathsf{SE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, and each of the algorithms in the tuple works as follows.

- $\mathsf{Gen}(1^\lambda) \rightarrow \mathsf{k}$: outputs a key k taking a security parameter 1^λ as input.
- $\mathsf{Enc}(\mathsf{k}, \mathsf{m}) \rightarrow \mathsf{ct}$: returns a ciphertext ct by encrypting a message m on symmetric key k.
- $\mathsf{Dec}(\mathsf{k}, \mathsf{ct}) \rightarrow \mathsf{m}$: takes a ciphertext ct and a symmetric key k as inputs and outputs a plaintext m.

The symmetric encryption scheme SE ensures indistinguishability under chosen-plaintext attack (IND-CPA) security and key indistinguishability under chosen-plaintext attack (IK-CPA [5]) security.

A.2 Public-key encryption

The public-key encryption scheme we use consists of a tuple of algorithms $\mathsf{PKE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ and works as follows.

- $\mathsf{Gen}(1^\lambda) \to (\mathsf{sk}, \mathsf{pk})$: returns a key pair $(\mathsf{sk}, \mathsf{pk})$ for secret key sk and public key pk taking a security parameter 1^λ as input.
- $\mathsf{Enc}(\mathsf{pk}, \mathsf{m}) \to \mathsf{ct}$: inputs a public key pk and a plaintext m, and outputs a ciphertext ct
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to \mathsf{m}$: takes a secret key sk and a ciphertext ct and outputs a plaintext m.

The encryption scheme PKE guarantees ciphertext indistinguishability under chosen-plaintext attack (IND-CPA) security and key indistinguishability under chosen-plaintext attack (IK-CPA [5]) security.

A.3 SNARK

A SNARK has to be complete, knowledge-sound, and succinct. A SNARK is complete if $\mathsf{Verify}(\mathsf{crs}, \mathbf{x}, \pi)$ outputs 1 with overwhelming probability for $(\mathbf{x}; \mathbf{w}) \in R$ and for any $\lambda \in \mathbb{N}$ and $R \in R_\lambda$ where $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, R)$ and $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, \mathbf{x}, \mathbf{w})$. Knowledge soundness (informally) means that a prover knows and can extract witness $\mathbf{w}$ from a proof π which passes the verification. For the succinctness, it means that the proof size and the verification time are logarithmic on the size of the witness. A SNARK may satisfy zero knowledge when nothing about the witness is leaked from the proof. We refer such SNARK to zk-SNARK and it can be constructed with the simulator which outputs a valid proof without knowing the witness $\mathbf{w}$.

B Notations for anonymous transfer (Azeroth)

Notation	**Description**
addr	User's address
EOA	Externally owned public account
ENA	Encrypted account
$\mathsf{k}^{\mathsf{ENA}}$	Symmetric key for encrypted account ENA
sct	Encrypted balance in ENA
$\mathsf{cm}_{\mathsf{Azeroth}}$	Commitment in the Azeroth protocol
$\mathsf{o}_{\mathsf{Azeroth}}$	Opening value of $\mathsf{cm}_{\mathsf{Azeroth}}$
$\mathsf{MT}_{\mathsf{Azeroth}}$	Merkle tree in the Azeroth protocol

Table 2: Notations related to Azeroth used in zkMarket

C Security analysis

In this section, we present the intuition of the security properties that zkMarket satisfies, as outlined in Section 3.1.

Fairness In the $\mathsf{RegisterData}$ phase, the seller registers the data using the hash values of the encrypted data (ct) and the symmetric encryption key (k). To obtain the k, the buyer must pay the purchase fee during the $\mathsf{GenerateTrade}$ phase, which then allows them to receive the k in the $\mathsf{AcceptTrade}$ phase, thereby ensuring seller-fairness.

For the seller to receive the fee for the data, the k must be delivered to the buyer in the $\mathsf{AcceptTrade}$ phase. The smart contract verifies the proof (π_{acc}) that the key has been encrypted with the buyer's public key, ensuring that the seller cannot receive the fee without providing the key to the buyer, thereby ensuring buyer-fairness.

Trade anonymity To prevent an eavesdropper obtain any details about the transaction, we use zk-SNARK, public key encryption, and Merkle trees. Information that eavesdroppers can access in zkMarket transactions includes $\mathsf{cm}, \mathsf{ct}_{\mathsf{order}}, \mathsf{sct}_{\mathsf{old}}, \mathsf{sct}_{\mathsf{new}}$ during the $\mathsf{GenerateTrade}$ phase, and $\mathsf{rt}, \mathsf{nf}, \mathsf{cm}_{\mathsf{Azeroth}}, \mathsf{h}_{\mathsf{k}}, \mathsf{ct}_{\mathsf{k}}, \mathsf{pk}^{\mathsf{seller}}, \mathsf{addr}^{\mathsf{seller}}$ during the $\mathsf{AcceptTrade}$ phase.

In the $\mathsf{GenerateTrade}$ phase, commitment schemes and public key encryption ensure that the eavesdropper cannot identify the specific content the buyer intends to purchase. Similarly, in the $\mathsf{AcceptTrade}$ phase, the eavesdropper cannot determine to whom the seller is delivering the content.

D Relations for zkMarket

As mentioned previously in Section 3.3, there exists three relations for each phase of zkMarket. Here we provide details of respective relations.

D.1 Relation for RegisterData

The relation for the registration phase R_{reg} consists of two parts, R_{ct} and $R_{\mathsf{h_k}}$.

Before explaining R_{ct}, we describe the relation for MatPRG, R_{MatPRG}, which is employed to encryption:

$$R_{\mathsf{MatPRG}}(\mathbf{A}; \mathbf{K}, \mathbf{R}, \gamma) = 1 \Leftrightarrow \mathbf{A} \times \mathbf{K} \times \gamma = \mathbf{R} \times \gamma$$

$R_{\mathsf{Matrix}\ \mathbf{R}}$ proves whether $\mathbf{R}$ is generated by $f_{\mathbf{A}}(\mathbf{K})$. To verify this efficiently, instead of directly checking $\mathbf{A} \times \mathbf{K} = \mathbf{R}$, the verification is performed using a random matrix γ by checking $\mathbf{A} \times \mathbf{K} \times \gamma = \mathbf{R} \times \gamma$.

Finally, R_{ct} is as follows:

$$R_{\mathsf{ct}}(\mathbf{A}; \mathsf{ct}, \mathsf{data}, \mathbf{K}, \mathbf{R}, \gamma) = 1 \Leftrightarrow R_{\mathsf{MatPRG}}(\mathbf{A}; \mathbf{K}, \mathbf{R}, \gamma) = 1 \wedge \Leftrightarrow \mathsf{ct} = \mathsf{data} + \mathsf{r}_{i,j},$$

The relation for hash R_{h} is as follows:

$$R_{\mathsf{h}}(\mathsf{h}; m) = 1 \Leftrightarrow \mathsf{h} = \mathsf{CRH}(m)$$

R_{h} proves that the hash value h is computed from the input m. Note that the multiple inputs are passed to hash function as concatenated (e.g., $\mathsf{CRH}(m_1 || m_2)$).

In conclusion, the relation for the registration phase, R_{reg}, is as follows:

$$R_{\mathsf{reg}}(\mathbf{A}, \mathsf{h}_{\mathsf{k}}; \mathsf{ct}, \mathsf{data}, \mathbf{K}, \mathbf{R}, \gamma, \mathsf{sk}^{\mathsf{seller}}) = 1 \Leftrightarrow$$

$$R_{\mathsf{h}}(\mathsf{h}_{\mathsf{k}}; \mathsf{sk}^{\mathsf{seller}}, \mathsf{k}) = 1 \wedge R_{\mathsf{ct}}(\mathbf{A}; \mathsf{ct}, \mathsf{data}, \mathbf{K}, \mathbf{R}, \gamma) = 1$$

R_{reg}, satisfies both R_{h} and R_{ct} at the same time, can prove the registration is correctly done without revealing the data before the deal is done. Based on the R_{reg}, the algorithm for register data is depicted in algorithm 2.

D.2 Relation for GenerateTrade

The relation for trade generation is then as follows:

$$R_{\text{com}}(\mathsf{cm}; m, r) = 1 \Leftrightarrow \mathsf{cm} = \mathsf{Com}(m; r)$$

$$R_{\text{PKE}}(\mathsf{pk}, \mathsf{ct}; m) = 1 \Leftrightarrow \mathsf{ct} = \mathsf{PKE.Enc}(\mathsf{pk}, m)$$

R_{com} proves that the commitment cm indeed commits to the message m with the randomness r. R_{PKE} proves that the ciphertext ct is genuinely from public-key encryption of the message m with the public key pk.

$$R_{\text{fee}}(\mathsf{sct}_{\text{old}}, \mathsf{sct}_{\text{new}}; \mathsf{k}^{\text{ENA}}, \mathsf{fee}) = 1 \Leftrightarrow \mathsf{fee} = \mathsf{SE.Dec}(\mathsf{k}^{\text{ENA}}, \mathsf{sct}_{\text{old}}) - \mathsf{SE.Dec}(\mathsf{k}^{\text{ENA}}, \mathsf{sct}_{\text{new}})$$

R_{fee} proves the buyer's ability to pay for the data. Namely, it checks that the payment fee is equal to the difference between the existing balance and the balance after payment. Since the account is encrypted for anonymous transfer, R_{fee} is proven under SNARK circuit by decrypting $\mathsf{sct}_{\text{old}}$ and $\mathsf{sct}_{\text{new}}$ with the decryption key k^{ENA}.

$$R_{\text{gen}}\begin{pmatrix} \mathsf{cm} & \mathsf{ct}_{\text{order}} \\ \mathsf{sct}_{\text{new}} & \mathsf{sct}_{\text{old}} \end{pmatrix}; \begin{matrix} r & \mathsf{h_k} & \mathsf{pk}^{\text{seller}} \\ \mathsf{pk}^{\text{buyer}} & \mathsf{k}^{\text{ENA}} & \mathsf{fee} \end{matrix}\Bigg) = 1$$

$$\Leftrightarrow R_{\text{com}}(\mathsf{cm}; \mathsf{pk}^{\text{seller}}, \mathsf{fee}, \mathsf{h_k}, \mathsf{pk}^{\text{buyer}}, r) = 1$$

$$\wedge R_{\text{PKE}}(\mathsf{pk}^{\text{seller}}, \mathsf{ct}_{\text{order}}; \mathsf{order}) = 1 \wedge R_{\text{fee}}(\mathsf{sct}_{\text{old}}, \mathsf{sct}_{\text{new}}; \mathsf{k}^{\text{ENA}}, \mathsf{fee}) = 1$$

At last, R_{gen} proves that: 1) cm is the commitment to fee with the public key of both buyer and seller, and the hash value of the symmetric key $\mathsf{h_k}$ over randomness r, 2) $\mathsf{ct}_{\text{order}}$ is an encryption of order with the public key of the seller $\mathsf{pk}^{\text{seller}}$, and 3) the payment fee is equal to the subtraction the decryption of $\mathsf{sct}_{\text{new}}$ from the decryption of $\mathsf{sct}_{\text{old}}$. Note that the decryption of sct is the balance of account.

D.3 Relation for AcceptTrade

Before demonstrating the relation for the trade acceptance phase, we provide the explanation for the Merkle tree relation since it is used in the acceptance phase.

$$R_{\text{MT}}(\mathsf{rt}; \mathsf{leaf}, \mathsf{Path}) = 1 \Leftrightarrow \mathsf{MT.MemVerify}(\mathsf{rt}, \mathsf{leaf}, \mathsf{Path}) = 1$$

R_{MT} proves that the given path (Path) is the authentication path of the leaf nodes leaf reaching to the root rt.

Finally, the relation for the trade acceptance phase R_{acc} is as follows:

$$R_{\text{acc}}\left(\begin{matrix} \mathsf{rt} & \mathsf{nf} \\ \mathsf{cm}_{\text{Azeroth}} & \mathsf{h_k} \\ \mathsf{ct_k} & \mathsf{pk}^{\text{seller}} \\ \mathsf{addr}^{\text{seller}} & \end{matrix}; \begin{matrix} \mathsf{cm} & \mathsf{Path} & \mathsf{sk}^{\text{seller}} \\ \mathsf{k} & \mathsf{pk}^{\text{buyer}} & r \\ \mathsf{fee} & \mathsf{o}_{\text{Azeroth}} & \end{matrix}\right) = 1$$

$$\Leftrightarrow R_{\mathsf{PKE}}(\mathsf{ct_k}; \mathsf{pk}^{\mathsf{buyer}}, \mathsf{k}) = 1 \wedge R_{\mathsf{h}}(\mathsf{h_k}; \mathsf{sk}^{\mathsf{seller}}, \mathsf{k}) = 1$$

$$\wedge R_{\mathsf{h}}(\mathsf{nf}; \mathsf{cm}, \mathsf{sk}) = 1$$

$$\wedge R_{\mathsf{com}}(\mathsf{cm}; \mathsf{pk}^{\mathsf{seller}}, \mathsf{fee}, \mathsf{h_k}, \mathsf{pk}^{\mathsf{buyer}}, r) = 1$$

$$\wedge R_{\mathsf{com}}(\mathsf{cm_{Azeroth}}; \mathsf{fee}, \mathsf{addr}^{\mathsf{seller}}, \mathsf{o_{Azeroth}}) = 1$$

$$\wedge R_{\mathsf{MT}}(\mathsf{rt}; \mathsf{cm}, \mathsf{Path}) = 1 \wedge R_{\mathsf{nf}}(\mathsf{nf}; \mathsf{cm}, \mathsf{sk}) = 1$$

R_{acc} contains six relations R_{PKE}, two different R_{h}, two different R_{com}, and R_{MT}. As explained previously, R_{PKE} proves that the $\mathsf{ct_k}$ is genuinely the encryption of the k which can be used to decrypt the ct with $\mathsf{pk}^{\mathsf{buyer}}$, the public key of buyer. The $\mathsf{h_k}$ is the hash value of the k and $\mathsf{sk}^{\mathsf{seller}}$ can be proven with the former R_{h}. Also, the other R_{h} proves that the nullifier nf is computed correctly with the commitment cm and the secret key of the seller sk. One of the R_{com} proves that cm is the commitment to fee with the public key of both buyer and seller, and $\mathsf{h_k}$ over randomness r. The other R_{com} is required for anonymous transfer, Azeroth, and it proves that $\mathsf{cm_{Azeroth}}$ commits to fee with the address of the seller.

E Deferred things for MatPRG

As described in Section 3.2, we devise new matrix-formed pseudorandom generator, MatPRG, to alleviate the proving overhead. In this section, we show deferred details related to MatPRG.

E.1 Pseudorandom Generator (PRG)

Informally, a pseudorandom generator produces a (long) sequence of numbers appearing randomly on a secret seed. The produced sequence should be computationally indistinguishable from a genuine random sequence.

Definition 4 (PRG, Pseudorandom Generator). *Let function $G : \{0,1\}^n \to \{0,1\}^m$ with $m > n$ is pseudo-random generator. Then, for all PPT adversaries $\mathcal{A}$ and random $y \in \{0,1\}^m$ and pseudo-random $G(x)$ for a random seed $x \in \{0,1\}^n$, there is a negligible function* negl *such that*

$$|\Pr[\mathcal{A}(1^n, y) = 1] - \Pr[\mathcal{A}(1^n, G(x)) = 1]| \leq \mathsf{negl}(n)$$

E.2 Linear System Model (LSM)

Let $m > n$. For a given matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, $\mathbf{K_2} \in \{0,1\}^{(m-x)\times k}$, and $\mathbf{R} \in \mathbb{Z}_q^{n\times k}$, the *linear system* algorithm finds the matrix $\mathbf{K_1} \in \{0,1\}^{x\times k}$ such that $\mathbf{A}\begin{bmatrix}\mathbf{K_1}\\ \mathbf{K_2}\end{bmatrix} = \mathbf{R}$, as follows:

1. Decompose $\mathbf{A}$ into an $n \times n$ square matrix $\mathbf{A_1}$ and an $n \times (m-n)$ matrix $\mathbf{A_2}$.
2. Compute the inverse of $\mathbf{A_1}$ (in this case, the matrix $\mathbf{A_1}$ is invertible with probability $1 - \frac{1}{q}$.) and multiply it by $\mathbf{R} - \mathbf{A_2}\mathbf{K_2}$:

$$\mathbf{K_1} = \mathbf{A_1}^{-1}(\mathbf{R} - \mathbf{A_2}\mathbf{K_2})$$

E.3 Security proof for MatPRG

Here, we present the security proof for the theorem 1 under the LSM defined in E.2.

Proof. Consider a matrix $\mathbf{A} = [\mathbf{A_1}\ \mathbf{A_2}] \in \mathbb{Z}_q^{n\times m}$, where $\mathbf{A_1} \in \mathbb{Z}_q^{n\times n}$ and $\mathbf{A_2} \in \mathbb{Z}_q^{n\times \delta}$, with $\mathbf{A_1}$ being an invertible matrix.

In the LSM, this adversary can construct $\mathbf{K_2}^*$ by randomly sampling from $\{0,1\}^{\delta\times k}$ and then compute $\mathbf{K_1}^* = \mathbf{A_1}^{-1}(\mathbf{R} - \mathbf{A_2}\mathbf{K_2^*})$. The adversary's objective is to distinguish between a truly pseudorandom output and a genuinely random one. To determine this, the adversary checks that $\mathbf{K_1}^* \in \{0,1\}^{x\times k}$ and scrutinizes whether $\mathbf{R} = f_{\mathbf{A}}(K^*)$. If the equation holds true, the adversary leans towards identifying the output as "pseudo-random." Conversely, if $\mathbf{R} \neq \mathbf{AK}^*$, the attacker inclines towards labeling the output as "random."

In essence, the adversary's success in distinguishing between the two outcomes relies on the uniqueness of $\mathbf{K_2}$. If the adversary's guess aligns with the true $\mathbf{K}$, indicating a pseudorandom output, the success probability is $\frac{1}{2^{\delta k}}$. Therefore, if $\delta k > 128$, the success probability becomes negligible, ensuring the security of our PRG.

F Algorithmic overview

According to the construction of zkMarket in Section 3.3, we depict the overall procedure of zkMarket in Figure 3.

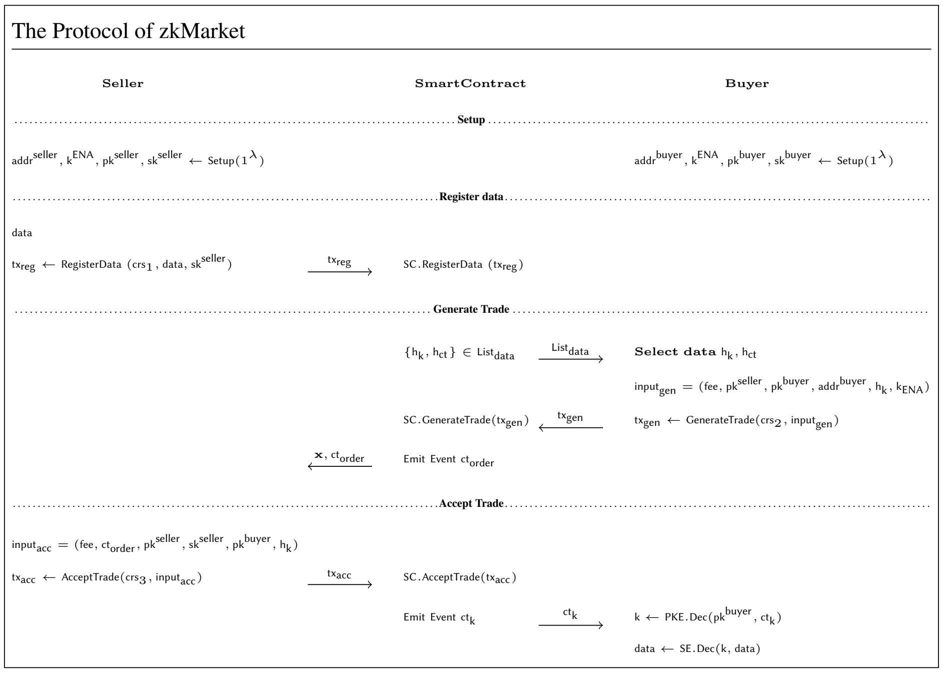
The Protocol of zkMarket
Seller
SmartContract
Buyer
Setup
$\mathsf{addr}^{\mathsf{seller}}, \mathsf{k}^{\mathsf{ENA}}, \mathsf{pk}^{\mathsf{seller}}, \mathsf{sk}^{\mathsf{seller}} \leftarrow \mathsf{Setup}(1^\lambda)$
$\mathsf{addr}^{\mathsf{buyer}}, \mathsf{k}^{\mathsf{ENA}}, \mathsf{pk}^{\mathsf{buyer}}, \mathsf{sk}^{\mathsf{buyer}} \leftarrow \mathsf{Setup}(1^\lambda)$
Register data
data
$\mathsf{tx_{reg}} \leftarrow \mathsf{RegisterData}\ (\mathsf{crs_1}, \mathsf{data}, \mathsf{sk}^{\mathsf{seller}})$
$\xrightarrow{\mathsf{tx_{reg}}}$
$\mathsf{SC.RegisterData}\ (\mathsf{tx_{reg}})$
Generate Trade
$\{\mathsf{h_k}, \mathsf{h_{ct}}\} \in \mathsf{List_{data}}$
$\xrightarrow{\mathsf{List_{data}}}$
Select data $\mathsf{h_k}, \mathsf{h_{ct}}$
$\mathsf{input_{gen}} = (\mathsf{fee}, \mathsf{pk}^{\mathsf{seller}}, \mathsf{pk}^{\mathsf{buyer}}, \mathsf{addr}^{\mathsf{buyer}}, \mathsf{h_k}, \mathsf{k_{ENA}})$
$\mathsf{SC.GenerateTrade}(\mathsf{tx_{gen}})$
$\xleftarrow{\mathsf{tx_{gen}}}$
$\mathsf{tx_{gen}} \leftarrow \mathsf{GenerateTrade}(\mathsf{crs_2}, \mathsf{input_{gen}})$
$\xleftarrow{\mathbf{x},\ \mathsf{ct_{order}}}$
Emit Event $\mathsf{ct_{order}}$
Accept Trade
$\mathsf{input_{acc}} = (\mathsf{fee}, \mathsf{ct_{order}}, \mathsf{pk}^{\mathsf{seller}}, \mathsf{sk}^{\mathsf{seller}}, \mathsf{pk}^{\mathsf{buyer}}, \mathsf{h_k})$
$\mathsf{tx_{acc}} \leftarrow \mathsf{AcceptTrade}(\mathsf{crs_3}, \mathsf{input_{acc}})$
$\xrightarrow{\mathsf{tx_{acc}}}$
$\mathsf{SC.AcceptTrade}(\mathsf{tx_{acc}})$
Emit Event $\mathsf{ct_k}$
$\xrightarrow{\mathsf{ct_k}}$
$\mathsf{k} \leftarrow \mathsf{PKE.Dec}(\mathsf{pk}^{\mathsf{buyer}}, \mathsf{ct_k})$
$\mathsf{data} \leftarrow \mathsf{SE.Dec}(\mathsf{k}, \mathsf{data})$

Fig. 3: Illustration of zkMarket workflow following algorithm in Section 3.3