# zkMarket: Ensuring Fairness and Privacy in Decentralized Data Exchange

Seongho Park*, Seungwoo Kim†, Semin Han*, Kyeongtae Lee*, Jihye Kim†‡, Hyunok Oh*†
*Hanyang University, Seoul, Republic of Korea
Email: {seonghopark, seminhan, rsias9049, hoh}@hanyang.ac.kr
†Kookmin University, Seoul, Republic of Korea
Email: {20172234, jihyek}@kookmin.ac.kr
‡Zkrypto Inc., Seoul, Republic of Korea

*Abstract*—**Ensuring fairness in blockchain-based data trading presents significant challenges, as the transparency of blockchain can expose sensitive details and compromise fairness. Fairness ensures that the seller receives payment only if they provide the correct data, and the buyer gains access to the data only after making the payment. Existing approaches face limitations in efficiency, particularly when applied to large-scale data. Moreover, preserving privacy has also been a significant challenge in blockchain.**

**In this paper, we introduce zkMarket, a privacy-preserving fair trade system on the blockchain. We ensure fairness by integrating encryption with zk-SNARKs, enabling verifiable proofs for fair trading. However, applying zk-SNARKs directly can be computationally expensive for the prover. To address this, we improve efficiency by leveraging our novel matrix-formed PRG (MatPRG) and commit-and-prove SNARK (CP-SNARK), making the data registration process more concise and significantly reducing the seller's proving time. To ensure transaction privacy, zkMarket is built upon an anonymous transfer protocol.**

**Experimental results demonstrate that zkMarket significantly reduces the computational overhead associated with traditional blockchain solutions while maintaining robust security and privacy. Specifically, our evaluation quantifies this high efficiency: the seller can register 1MB of data in 2.8 seconds, the buyer can generate the trade transaction in 0.2 seconds, and the seller can finalize the trade within 0.4 seconds.**

## I. 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 [1]. Under this model, various approaches depending on a TTP have been proposed [2]–[4]. However, with the emergence of blockchain [5], [6] with the advantages of immutability and transparency, along with smart contracts, it has garnered significant attention. A line of work has proposed blockchain-based fair trade protocols that eliminate the need for a TTP [7]–[13].

Despite these advancements, designing a fair blockchain-based data trading system remains a significant challenge, particularly concerning the transparency of transactions. While this transparency is a core blockchain advantage, it paradoxically allows individuals who have not paid to potentially access data, compromising fairness. The prevailing approach to construct fair data exchange protocols on blockchains involves a Hash Time Lock Contract (HTLC) [14] combined with a non-interactive zero-knowledge proof (NIZK) [15]–[18]. This intuitive method has sellers encrypting data, trading the decryption key, and proving key correctness via NIZK, with payment released upon proof verification and key revelation.

The recent work SmartZKCP [19] highlights that the seller with extensive computational costs is potentially exposed 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, it presents challenges for real-world scenarios, especially when scaling to large data sizes.

Addressing the general problem of large-scale data, Tas et al. propose a fair data exchange protocol for large-scale data on blockchain [10]. Due to the practical limitations of on-chain storage costs on blockchains, authors assume the ciphertext of the data is publicly accessible. Thus, on-chain trading involves only the decryption key for the ciphertext of the data. Instead of relying on the standard HTLC and NIZK approach, their work introduces a novel cryptographic concept called Verifiable Encryption under Committed Key (VECK) to ensure that the buyer receives the correct data. VECK allows the ciphertext to be verified using a verification key, which is a commitment to the decryption key. VECK can prove the validity of large-sized data; however, the verification time increases proportionally to the data size.

Beyond fairness, privacy stands as another critical consideration in blockchain-based data exchange. The public nature of blockchain transactions enables adversaries to analyze transaction histories and infer sensitive private details about individuals or organizations, potentially leading to exploitation. While protocols like Avizheh et al. [20] have proposed privacy-preserving data exchange using circuit randomization to shield transaction details, this technique demands extremely high seller runtime—for example, approximately 424 seconds to encode just 66KB of data[1]—posing a significant challenge for real-world applications.

We introduce zkMarket, a privacy-preserving fair digital data trading system on blockchain, which aims to fill this critical gap and operate effectively under such practical conditions.

zkMarket ensures fairness by leveraging a zero-knowledge succinct non-interactive argument of knowledge (zk-SNARK). Specifically, the seller publishes commitments to the ciphertext of the data (and the decryption key) to prevent a malicious buyer from obtaining the data without payment or a malicious seller from manipulating the data. The seller then submits a zk-SNARK proof demonstrating that the encrypted data is indeed the ciphertext of the data the buyer requested to purchase and that the commitments are validly computed. The buyer locks the payment in the smart contract, and the seller gains the locked fee if and only if the zk-SNARK proof is verified. This guarantees that the seller can claim the fee only if they provide the correct data, and the buyer can receive the data only if they pay the fee. However, encoding encryption and commitment within the zk-SNARK circuit imposes significant costs on the prover.

Furthermore, zkMarket enhances privacy through anonymous trading. Our system validates transaction integrity using encrypted accounts and zk-SNARKs, ensuring sensitive details such as the identities of the buyer and seller, the nature of the traded data, or the payment amount, remain confidential to all but the trading participants, even while transactions are publicly verifiable. We integrate an existing auditable anonymous transfer technique [21] to safeguard financial details, providing a crucial balance between privacy and accountability for real-world applications.

To make zkMarket more efficient, even for large-sized data, we introduce a novel technique called matrix-formed pseudorandom generator (MatPRG). zkMarket uses the counter (CTR) mode for symmetric key encryption to encrypt data. Conventionally, proving such encryption within the zk-SNARK circuit requires embedding SNARK-friendly hash functions such as MiMC [22] or Poseidon [23]. However, even these optimized hashes incur significant constraint costs—often hundreds per block—especially when encrypting large-scale datasets.

To overcome this bottleneck, MatPRG replaces iterative hashing with a single matrix-vector multiplication over a prime field. This approach preserves pseudorandomness while drastically reducing proving cost, cutting hundreds of constraints per block down to just a few. To the best of our knowledge, MatPRG is the first matrix-based pseudorandom generator primitive. Additionally, we leverage the commit-and-prove SNARK (CP-SNARK) framework [24] to move commitment computation outside the SNARK circuit, further reducing the circuit complexity.

Our evaluation highlights substantial improvements in proving time for 1MB of data. With our optimizations, the proving time is drastically reduced to 2.8 seconds, achieving roughly a 367x speedup over the unoptimized baseline (1030 seconds). Furthermore, this is approximately 250 times faster than the VECK approach (703.6 seconds) for the identical data size.

[1]The estimation is based on the experimental results presented in [20].

### A. Our contributions

Our contributions are as follows:

- We propose zkMarket, which is a blockchain-based digital data trade system providing fairness through combining encryption and zk-SNARK. zkMarket also enables the participants to trade anonymously by employing an anonymous transfer protocol and is robust against DoS attacks.
- We also significantly reduce the proving cost of the seller incurred during the data registration by employing 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 2.8 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.

## II. Related work

The Zero-Knowledge Contingent Payment (ZKCP) [25] protocol leverages 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 [26] zk-SNARK, which needs a trusted setup, ZKCPlus [27] 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 a 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 [19] 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.

## III. Preliminaries

We introduce the notations and (informal) definitions of cryptographic primitives used throughout this paper. We denote $x \leftarrow\$ \mathbb{F}$ for sampling the random $x$ from the finite field $\mathbb{F}$ of size $q$. A hash function denoted as $\mathsf{CRH}$ is a collision-resistant hash function. A matrix $\mathbf{A} \in \mathbb{F}^{n \times m}$ indicates a matrix with $n$ rows and $m$ columns where all elements are drawn from $\mathbb{F}$. A matrix can also be presented in binary. Namely, $\mathbf{A} \in \mathbb{F}^{n \times m}$ is identical to $\mathbf{A} \in \{0,1\}^{n \times m \times \log q}$. All formal definitions are described in Appendix A.

### *A. 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 [28]) security.

### *B. Commitment*

We use a standard commitment scheme $\mathsf{COM} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{VerifyCom})$ where $\mathsf{Setup}(1^\lambda)$ outputs a commitment key $\mathsf{ck}$ taking the security parameter as input, $\mathsf{Com}(\mathsf{ck}, m)$ returns the commitment $c$ and the opening randomness $o$ where message $m$ is committed with $\mathsf{ck}$, and $\mathsf{VerifyCom}(\mathsf{ck}, c, m, o)$ takes as input a commitment, message, and opening, and accepts if a commitment is valid. 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 verified with the original committed value.

### *C. Merkle Tree*

Merkle Tree is a data structure where a party can commit to some value succinctly and further prove membership of some leaf value. Briefly, each leaf is computed by (collision-resistant) hashing a 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 $\mathsf{rt}$. The path for proving membership of a specific node (nd) is denoted as $\mathsf{path}_{\mathsf{nd}}$. Here is a brief description of the syntax in this work: $\mathsf{ComputePath}(\mathsf{nd}) \rightarrow \mathsf{path}$ produces an authentication path $\mathsf{path}$ to the $\mathsf{rt}$; $\mathsf{MemVerify}(\mathsf{rt}, \mathsf{nd}, \mathsf{path}_{\mathsf{nd}}) \rightarrow 0/1$ returns 1 if the membership proof $\mathsf{path}_{\mathsf{nd}}$ is valid against the root $\mathsf{rt}$ and a leaf node $\mathsf{nd}$, 0 otherwise; $\mathsf{TreeUpdate}(\mathsf{nd}_{\mathsf{new}}) \rightarrow \mathsf{rt}_{\mathsf{new}}$ returns new root $\mathsf{rt}_{\mathsf{new}}$ for the updated Merkle Tree on newly added value $\mathsf{nd}_{\mathsf{new}}$.

### *D. Succinct Non-interactive ARguments of Knowledge (SNARKs)*

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

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

If SNARK is zero-knowledge, we refer to it as a zk-SNARK. A commit-and-prove SNARK (CP-SNARK) [24] is a SNARK that integrates commitment schemes into the proof system to efficiently prove properties of committed inputs, enabling modular composition with committed witnesses. For instance, $\mathcal{R}_1(c, w_1)$ and $\mathcal{R}_2(c, w_2)$ can be proven through CP-SNARK relation $\mathcal{R}(c, w_1, w_2, m, o) : \mathcal{R}(c, w_1, w_2, m, o) = 1 \Leftrightarrow \mathcal{R}_1(c, w_1) = 1 \wedge \mathcal{R}_2(c, w_2)$.

### *E. Pseudorandom Generator (PRG)*

Informally, a pseudorandom generator produces a (long) sequence of numbers appearing random on a secret seed. For $m > n$, the produced sequence outputted by a pseudorandom generator $G : \{0,1\}^n \rightarrow \{0,1\}^m$ should be computationally indistinguishable from a genuine random sequence.

### *F. Anonymous transfer protocol*

To trade digital content on blockchain, a fee transaction inevitably occurs between the seller and the buyer. On blockchains like Ethereum, the transaction is public 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. For user privacy on a public blockchain, we apply the anonymous transfer protocol, such as Azeroth [21], Zerocash [29], and BlockMaze [30]. In this paper, we employ Azeroth due to its advantages in the efficiency of anonymous transfer and gas consumption and its support for anti-money laundering by providing auditability.

**Revisit Azeroth** Azeroth [21] is an auditable anonymous transfer protocol designed for smart contracts. It introduces two types of accounts: an externally owned account (EOA), which is publicly visible, and an encrypted account (ENA), whose balance is stored in an encrypted form denoted by sct. Each ENA is linked to an on-chain address $\mathtt{addr}$, and both account types operate on the blockchain with encrypted balances to ensure privacy. When a user initiates a fund transfer, a new commitment $\mathsf{cm}_{\mathsf{Azeroth}}$ is created and inserted into the Merkle tree $\mathsf{MT}_{\mathsf{Azeroth}}$, encapsulating the transaction details. This commitment includes the recipient's encrypted information, allowing only the intended recipient who possesses the correct decryption key to recognize and claim the transaction. Since the recipient's identity is concealed within the encrypted payload, external observers cannot determine the destination of funds, thereby achieving transaction confidentiality. Additionally, the inclusion of commitments in the Merkle Tree provides an audit trail, enabling verifiability without compromising anonymity.

## IV. zkMarket

### *A. Model and Security Assumptions*

In our system model, we define the core entities and operational phases of zkMarket. Figure 1 provides a brief

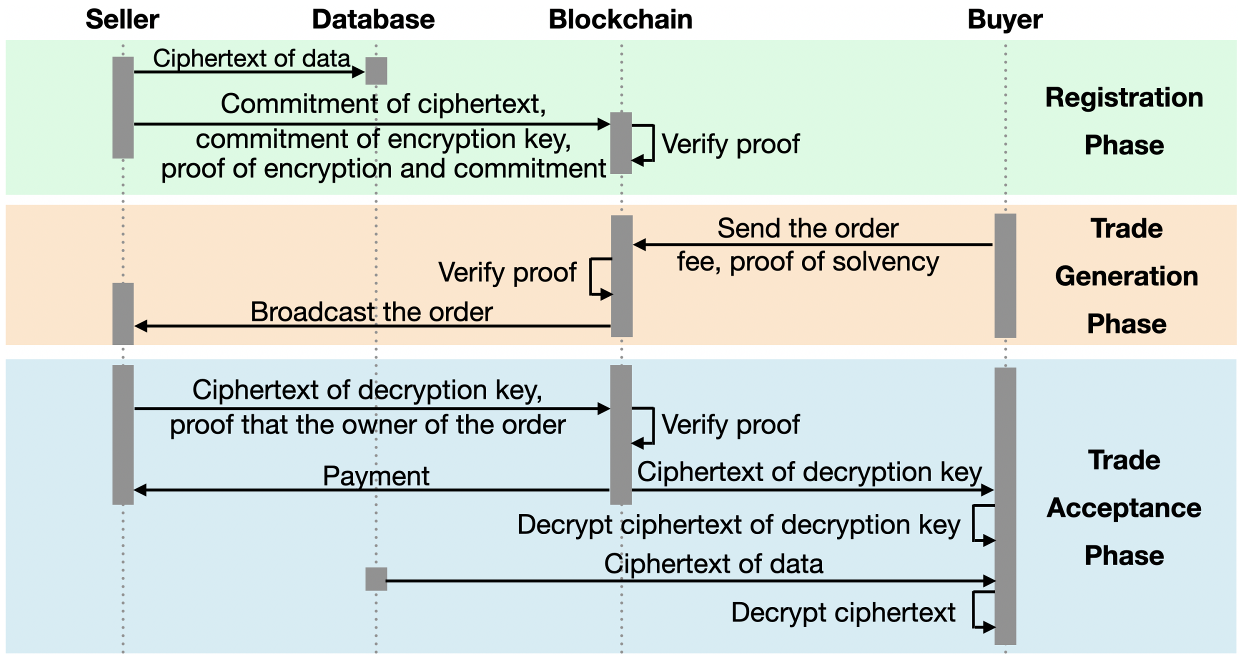

Seller
Database
Blockchain
Buyer
Ciphertext of data
Commitment of ciphertext, commitment of encryption key, proof of encryption and commitment
Verify proof
Registration Phase
Send the order fee, proof of solvency
Verify proof
Broadcast the order
Trade Generation Phase
Ciphertext of decryption key, proof that the owner of the order
Verify proof
Payment
Ciphertext of decryption key
Decrypt ciphertext of decryption key
Ciphertext of data
Decrypt ciphertext
Trade Acceptance Phase

Fig. 1: Sequence diagram of zkMarket

overview of zkMarket. The system involves three main types of entities: a data seller, a data buyer, and a blockchain.

- **Seller**: A seller is a data owner who intends to sell their encrypted data, such as medical records, documents, or datasets, on the market. A key aspect of the seller's participation is their ability to engage in trading without revealing their real-world identity or the plaintext content of the data being sold.
- **Buyer**: A buyer is an entity interested in purchasing data from a seller. Buyers initiate the trade process by submitting a trade request to the blockchain.
- **Blockchain**: The blockchain serves as the decentralized platform where the zkMarket smart contract is deployed and executed. This smart contract enforces the predetermined conditions for a trade to proceed and, upon their verification, automates the transfer of funds to ensure the atomicity of the transaction.

zkMarket consists of a setup phase followed by three main phases to facilitate a complete data trade: 1) The *Registration phase*, where the seller registers their encrypted data and associated information on the blockchain market; 2) The *Trade generation phase*, initiated by a buyer who requests to purchase specific registered data; and 3) The *Trade acceptance phase*, where the seller reviews and approves the purchase request submitted by the buyer.

TABLE I: Notations related to Azeroth used in zkMarket

| 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 |

Due to the practical limitations of on-chain storage costs in blockchain systems, we assume that the ciphertext of the data is publicly accessible. This approach is also taken by Tas et al. [10]. We do not address how the buyer determines the value or validity of the encrypted data; such decisions are assumed to be made externally, prior to engaging with our protocol. Accordingly, the trading mechanism focuses solely on the decryption key. Within this model, potential security threats arise primarily from the actions of malicious participants. A *malicious seller* might attempt to receive payment without genuinely providing the correct decryption key for the requested data. Conversely, a *malicious buyer* could try to obtain the decryption key (and subsequently the data) without fulfilling their payment obligation. Furthermore, the inherent transparency of public blockchains introduces a privacy challenge, as a *malicious third party* could potentially observe transaction details and infer sensitive information, such as the identities of trading parties or their purchasing patterns, without consent.

To mitigate these identified risks and prevent such malicious behaviors, zkMarket is designed to uphold the following critical security properties:

- **Fairness**: This property is divided into two sub-properties: *seller fairness* and *buyer fairness* [19]. Seller fairness guarantees that no buyer can access the data, in whole or in part, before they have successfully completed

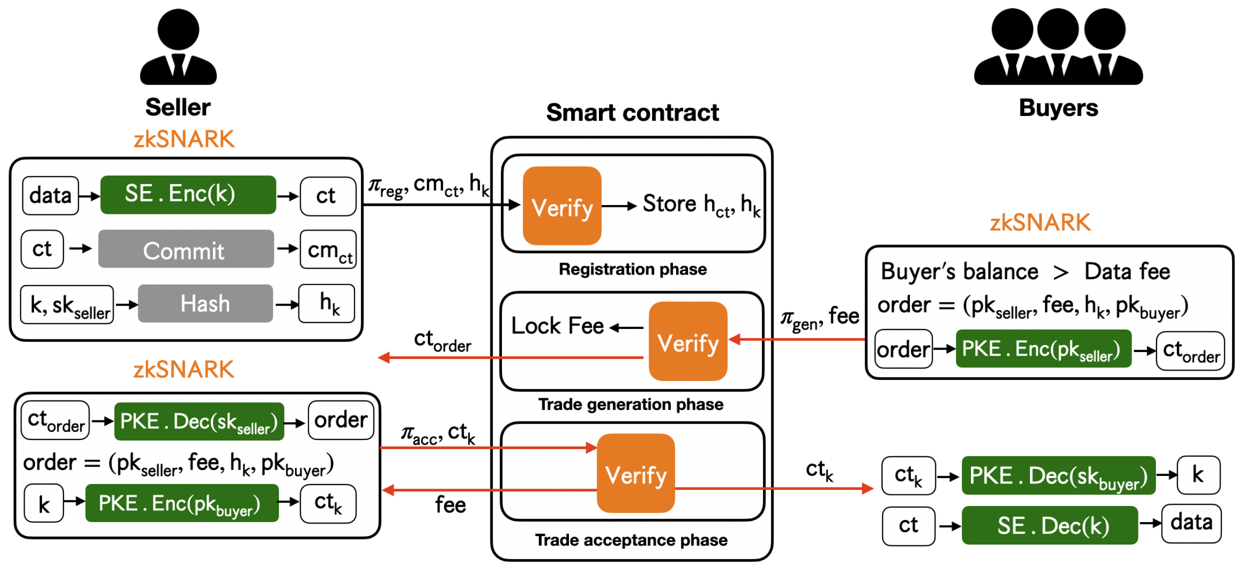


Fig. 2: The overview of zkMarket. Transactions depicted in red color represent the Azeroth transactions.

the required payment. Buyer fairness ensures that a seller cannot receive the payment unless they have genuinely delivered the decryption key corresponding to the data that the buyer intended to purchase.

- **Trade anonymity**: The protocol aims to provide anonymity, ensuring that no unauthorized party can link transaction details to real-world identities or determine which specific data was traded or the amount involved.
- **Denial-of-Service (DoS) attack**: We also consider the threat of DoS attacks, where an adversary aims to deplete the computational resources of honest participants. This could involve scenarios such as a malicious buyer repeatedly submitting frivolous trade requests or a malicious seller intentionally sending malformed or invalid ciphertexts that force buyers into costly and unproductive verification processes.

## *B. Construction*

The detailed algorithms are depicted in Algorithm 1 through Algorithm 4. Prefixed with SC denotes the algorithms executed on the smart contract. The setup phase generates a common reference string (CRS) for zk-SNARK for three relations $R_{\text{reg}}$, $R_{\text{gen}}$, and $R_{\text{acc}}$ for which corresponding to three main phases. We provide detailed descriptions of the relations in Appendix B, due to the space limit. It also generates the commitment key for a commitment scheme and invokes the setup of Azeroth for anonymous transfer. The smart contract stores the verification keys and initializes the Merkle tree to facilitate anonymous transactions.

**Registration phase.** In the registration phase, the seller encrypts the data. No one can access (or even deduce) the original data, and only the ciphertext of the data is publicly available. The seller submits both the commitment to the ciphertext ($\mathsf{cm}_{\mathsf{ct}} \leftarrow \mathsf{COM.Com}(\mathsf{ck}, \mathsf{ct}; o)$) and the hash values of the decryption key ($\mathsf{h}_{\mathsf{k}} \leftarrow \mathsf{CRH}(\mathsf{sk}^{\mathsf{seller}} \| \mathsf{k})$), along with its zk-SNARK proof to the smart contract. Note that the registration is required only once, thereby minimizing the prover's overhead.

**Trade generation phase.** During the trade generation phase (Algorithm 3), the buyer submits a trade request and demonstrates their ability to pay using zk-SNARK. The buyer commits to the payment fee to prevent a malicious seller from denying the purchase or decreasing the payment amount below the agreed price (i.e., to provide seller fairness). The buyer also encrypts the transaction details, including the payment. For anonymous transfer, 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 buyer produces the zk-SNARK proof $\pi_{\text{gen}}$ demonstrating that: 1) the commitment of the order is valid; 2) the ciphertext of the order is validly encrypted using $\mathsf{pk}^{\mathsf{seller}}$; and 3) the buyer has a sufficient balance to pay the fee.

The SC.GenerateTrade handles the buyer's request 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 $\pi_{\text{gen}}$. Once the proof verification is passed, it updates $\mathsf{sct}_{\mathsf{old}}$ to $\mathsf{sct}_{\mathsf{new}}$ and

**Algorithm 1** Setup phase

```
$\underline{\mathsf{Setup}(1^\lambda):}$
$\mathsf{crs}_1 \leftarrow \Pi_{\mathsf{snark}}.\mathsf{Setup}(1^\lambda, R_{\text{reg}})$
$\mathsf{crs}_2 \leftarrow \Pi_{\mathsf{snark}}.\mathsf{Setup}(1^\lambda, R_{\text{gen}})$
$\mathsf{crs}_3 \leftarrow \Pi_{\mathsf{snark}}.\mathsf{Setup}(1^\lambda, R_{\text{acc}})$
$\mathsf{ck} \leftarrow \mathsf{COM.Setup}(1^\lambda)$
$(\mathsf{addr}, \mathsf{k}^{\mathsf{ENA}}) \leftarrow \mathsf{Azeroth.Setup}(1^\lambda)$
return $\mathsf{crs} := (\mathsf{addr}, \mathsf{k}^{\mathsf{ENA}}, \mathsf{ck}, \mathsf{crs}_{reg}, \mathsf{crs}_{gen}, \mathsf{crs}_{acc})$
```

**Algorithm 2** Registration phase

```
Off − chain
RegisterData(crs, data, sk^{seller}) :
    $k \leftarrow \mathsf{SE.Gen}(1^\lambda)$
    $ct \leftarrow \mathsf{SE.Enc}(k, data)$
    $h_k \leftarrow \mathsf{CRH}(sk^{seller} || k)$
    $o \leftarrow\$ \mathbb{F}$
    $cm_{ct} \leftarrow \mathsf{COM.Com}(ck, ct; o)$
    $\mathbf{x} := (h_k, cm_{ct})$
    $\mathbf{w} := (ct, data, k, sk^{seller}, o)$
    $(\pi_{reg}) \leftarrow \Pi_{snark}.\mathsf{Prove}(crs_{reg}, ck, \mathbf{x}; \mathbf{w})$
    return $tx_{reg} = (h_k, cm_{ct}, \pi_{reg})$

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

**Algorithm 3** Trade generation phase

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

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

updates $\mathsf{cm}_{\mathsf{order}}$ to the Merkle tree $\mathsf{MT}$. During this process, the buyer's balance is updated to reflect the deduction of the purchase cost. Finally, it emits an event[2] with $\mathsf{ct}_{\mathsf{order}}$ to enable the seller to process the order.

**Trade acceptance phase.** In the trade acceptance phase (Algorithm 4), the seller approves the buyer's trade request and transmits the ciphertext of the decryption key via the blockchain. From the pool of encrypted trade requests submitted by buyers(events emitted in the trade generation phase), the seller decrypts a request $\mathsf{ct}_{\mathsf{order}}$ using their secret key, $\mathsf{ct}_{\mathsf{order}} \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}^{\mathsf{seller}}, \mathsf{ct}_{\mathsf{order}})$, to identify if it was addressed to them. Upon successful decryption, the seller retrieves the order details, including the buyer's public key. Then, the seller encrypts the decryption key using the buyer's public key ($\mathsf{ct}_{\mathsf{k}} \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}^{\mathsf{buyer}}, \mathsf{k})$). Since the seller shares the same values after decrypting $\mathsf{ct}_{\mathsf{order}}$, they can reconstruct the same commitment $\mathsf{cm}_{\mathsf{order}}$ originally created by the buyer. This allows the seller to verify that the order is included in the public order list and to generate a nullifier $\mathsf{nf}$ to mark the trade as completed, thereby preventing double-spending. Subsequently, the seller generates a new commitment $\mathsf{cm}_{\mathsf{Azeroth}}$ to claim the associated fee fee via an anonymous transfer using the Azeroth protocol. After verifying the zk-SNARK proof $\pi_{\mathsf{acc}}$, the commitment is inserted into the Azeroth Merkle tree $\mathsf{MT}_{\mathsf{Azeroth}}$. Once recorded, the seller can withdraw the fee at any future time through the Azeroth transfer mechanism.[3] Finally, the seller then sends $\mathsf{ct}_{\mathsf{k}}$—not the raw key—along with a zk-SNARK proof $\pi_{\mathsf{acc}}$ demonstrating that $\mathsf{ct}_{\mathsf{k}}$ correctly encrypts $\mathsf{k}$ under $\mathsf{pk}^{\mathsf{buyer}}$. This guarantees that only the intended buyer can decrypt and access the key. Furthermore, no information about the key is leaked through the transaction itself.

The zk-SNARK proof $\pi_{\mathsf{acc}}$ validates the following conditions: $\mathsf{ct}_{\mathsf{k}}$ is encryption of $\mathsf{k}$, $\mathsf{cm}_{\mathsf{Azeroth}}$ is correctly computed for anonymous transfer, and $\mathsf{cm}_{\mathsf{order}}$'s Merkle tree membership is proven. Additionally, $\pi_{\mathsf{acc}}$ ensures the transaction is processed only once by verifying $\mathsf{nf}$ is not in $\mathsf{List}_{\mathsf{data}\mathsf{nf}}$.

SC.AcceptTrade allows the seller to accept trade requests

[2]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.

[3]The commitment $\mathsf{cm}_{\mathsf{Azeroth}}$ stored in the Merkle tree can be transferred to the seller's encrypted account (ENA) using the zkTransfer algorithm in Azeroth. See [21] for details.

**Algorithm 4** Trade acceptance phase

```
Off – chain
AcceptTrade(crs, rt, path, fee, $ct_{order}$, $pk^{seller}$, $sk^{seller}$, $pk^{buyer}$, $h_k$) :
$order \leftarrow PKE.Dec(sk^{seller}, ct_{order})$
parse $order = (r, fee, h_k, pk^{buyer})$
$ct_k \leftarrow PKE.Enc(pk^{buyer}, k)$
$cm_{order} \leftarrow COM.Com(ck, pk^{seller}||fee||h_k||pk^{buyer}; r)$
$nf \leftarrow CRH(cm_{order}||sk^{seller})$
$o_{Azeroth} \leftarrow\$ \mathbb{F}$
$cm_{Azeroth} \leftarrow COM.Com(ck, fee||addr^{seller}; o_{Azeroth})$
$\mathbf{x} := (rt, nf, cm_{Azeroth}, h_k, ct_k, addr^{seller})$
$\mathbf{w} := (cm_{order}, path, sk^{seller}, k, pk^{seller}, pk^{buyer}, r, fee, o_{Azeroth})$
$\pi_{acc} \leftarrow \Pi_{snark}.Prove(crs_{acc}, \mathbf{x}; \mathbf{w})$
return $tx_{acc} = (\mathbf{x}, \pi_{acc})$

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

by verifying $\pi_{acc}$. Finally, nf is added to $List_{datanf}$, $MT_{Azeroth}$ is updated, and an event with $ct_k$ is emitted, enabling the buyer to retrieve the decryption key.

## C. Security analysis

**Theorem IV.1.** *Let* CRH *be a secure cryptographic hash function,* PKE *be an IND-CPA and IK-CPA secure public key encryption scheme,* SE *be a secure symmetric key encryption,* $\Pi_{snark}$ *be a secure commit-and-prove SNARK,* COM *ensures hiding and binding, and the security of Azeroth holds. The zkMarket guarantees fairness, trade anonymity, and security against DoS attacks.*

*Proof.* **Fairness.** As mentioned earlier, we prove two sub properties: *seller fairness* and *buyer fairness*.

*Seller fairness.* Suppose a malicious buyer receives the key $k$ without paying the fee. To do this, the buyer must obtain $ct_k$ without posting the required fee and generating a valid $\pi_{gen}$. But in zkMarket, the trade acceptance phase is executed only if the fee has already been locked via a valid $\pi_{gen}$. Thus, this contradicts the transaction logic enforced by the smart contract, together with SNARK soundness.

*Buyer fairness.* Suppose a malicious seller receives the fee without sending the correct key k. Then, there exists $ct_k$ and $\pi_{acc}$ such that $\pi_{acc}$ is accepted by the smart contract, and $ct_k \neq PKE.Enc(pk^{buyer}, k')$ for any $k'$ such that $SE.Dec(k', ct) = data$. This contradicts the soundness of the SNARK, which ensures that $\pi_{acc}$ is only accepted if $ct_k$ encrypts a valid k corresponding to the committed ct.

**Trade anonymity.** Let $\mathcal{A}$ be an adversary who tries to distinguish two orders or link a ciphertext $ct_k$ or $ct_{order}$ to a specific user. From the IND-CPA security of PKE, both $ct_k$ and $ct_{order}$ reveal no information about the underlying plaintexts. From the zero-knowledge property of zk-SNARK, $\pi_{reg}$, $\pi_{gen}$, and $\pi_{acc}$ reveal nothing about witness values such as k, order details, or public keys. The hiding property of COM ensures that $cm_{ct}$ does not reveal ct. Since all values posted to the smart contract are either commitments, ciphertexts, or zk-SNARK proofs, and none leak information, we conclude that $\mathcal{A}$ learns nothing about the participants or transaction content beyond what is publicly visible. Thus, any successful attempt by $\mathcal{A}$ to break anonymity would violate either IND-CPA security of SE or PKE or zero-knowledge of zk-SNARK or hiding property of COM.

**Denial of Service (DoS) attack.** There are three scenarios for a DoS attack. First, a malicious buyer posts an invalid order to waste the seller's resources. This can be prevented by requiring $\pi_{gen}$. By soundness of zk-SNARK, the probability of generating $\pi_{gen}$ using an invalid order is negligible. Second, a malicious seller posts an invalid $ct_k$ to block buyer access. This also can be prevented by the soundness of the zk-SNARK of $\pi_{acc}$, which must prove that $ct_k$ decrypts to the correct $k$. Lastly, repeated interactions to congest the smart contract can be mitigated by requiring deposits (fees), and invalid proofs are rejected without storage cost. Thus, under the soundness of zk-SNARK and the logic of the smart contract, DoS attacks are mitigated. □

## V. Analysis on Registration phase

Recalling the registration phase, the seller generates the zk-SNARK proof $\pi_{reg}$ establishing that: (1) the ciphertext ct is indeed the encryption of data, (2) $cm_{ct}$ is the correct commitment of ct, and (3) $h_k$ is the hash output of the encryption key k (used for data) and the seller's secret key $sk^{seller}$, i.e., $h_k = CRH(sk^{seller}||k)$. Generating $\pi_{reg}$ introduces efficiency bottlenecks that escalate with data size.

Even when proving the encryption efficiently within a zk-SNARK circuit, for instance, by employing block ciphers based on PRG using SNARK-friendly hash functions such as MiMC7 [22] or Poseidon [23], these methods still incur high costs. Generating a proof for registering 1MB-sized data requires approximately 1,000 seconds, as illustrated in Figure 3. Similarly, proving ciphertext commitment also involves costly in-circuit hash computations, adding significant overhead.

To tackle these challenges, we introduce two key optimizations. We propose a novel pseudorandom generator, MatPRG, which replaces numerous hash operations with a single matrix multiplication for encryption proof, drastically reducing proving costs. Additionally, we leverage the commit-and-prove SNARK (CP-SNARK) framework [24], allowing us to move commitment computation outside the zk-SNARK circuit and further reduce complexity. Combining MatPRG and CP-SNARK enhances the practicality, requiring 2.3s for proving 1MB of data.

## *A. MatPRG: Matrix-formed PRG*

MatPRG is defined under the *Decisional Short Integer Solution (DSIS)* [31]. A standard Short Integer Solution (SIS) problem is that given a uniformly random matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, the goal is to find a non-zero integer vector $\mathbf{k} \in \mathbb{Z}^m$ such that $\mathbf{A}\cdot\mathbf{k} = \mathbf{0} \mod q$ and $|\mathbf{k}| \leq \beta$ for some norm bound $\beta$. SIS is a foundational hard problem in lattice-based cryptography.

The Decisional Short Integer Solution (DSIS) problem is the decisional variant of the SIS problem. It asks whether a given vector was generated using a short integer solution or sampled uniformly at random. Let $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ be a public random matrix, and let $\mathbf{K} = \begin{bmatrix}\mathbf{K}_1\\ \mathbf{K}_2\end{bmatrix} \in \{0,1\}^{m\times l}$ be a binary secret key, where $\mathbf{K}_1 \in \{0,1\}^{n\times l}$ is chosen uniformly at random. MatPRG generates its pseudorandom output as $\mathbf{R} = \mathbf{A}\cdot\mathbf{K} \in \mathbb{Z}_q^{n\times l}$.

**Definition V.1** (Decisional Short Integer Solution Problem [31])**.** *Given a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ and a vector $\mathbf{R} \in \mathbb{Z}_q^n$, the advantage for distinguishing whether $\mathbf{R}$ was generated as $\mathbf{R} = \mathbf{A}\cdot\mathbf{K} \mod q$ for a short vector $\mathbf{K}$ with $|\mathbf{K}| \leq \beta$, or sampled uniformly at random from $\mathbb{Z}_q^{n\times m} \times \mathbb{Z}_q^n$ is negligible.*

**Definition V.2.** *Let $l, m, n, q \in \mathbb{N}$ with $q$ a prime and $m > n$ such that $m \approx n$. Let $\mathbf{A} \leftarrow\!\!\$\ \mathbb{Z}_q^{n\times m}$ be a uniformly random matrix over the finite field $\mathbb{Z}_q$, and let $\mathbf{K} \in \{0,1\}^{m\times l}$ be an input binary matrix. Define a function $f_{\mathbf{A}} : \{0,1\}^{m\times l} \to \{0,1\}^{n\times l\times \log q}$ by multiplying two matrices $\mathbf{A}\cdot\mathbf{K}$ and representing it as a binary form.*

**Theorem V.1.** *Let $f_{\mathbf{A}}$ be defined as in Definition V.2, with $\mathbf{A} \leftarrow\!\!\$\ \mathbb{Z}_q^{n\times m}$ uniformly at random, $m = n + \delta$ for $\delta > 0$, and $\delta \cdot l > \lambda$ for some security parameter $\lambda$.*

*Then, under the hardness of the DSIS problem, it is computationally hard to determine whether, for a given matrix $\mathbf{R} \in \mathbb{Z}_q^{n\times l}$, there exists $\mathbf{K} \in \{0,1\}^{m\times l}$ such that $f_{\mathbf{A}}(\mathbf{K}) = \mathbf{R}$. Furthermore, computing such a $\mathbf{K}$ (when it exists) is also computationally infeasible.*

*Proof.* Suppose there exists a PPT algorithm $\mathcal{A}$ that, given $(\mathbf{A}, \mathbf{R})$, decides whether there exists a binary matrix $\mathbf{K}$ such that $\mathbf{A}\cdot\mathbf{K} = \mathbf{R}$. Next, we construct a distinguisher $\mathcal{D}$ for the DSIS problem. Given $(\mathbf{A}, \mathbf{R})$, $\mathcal{D}$ runs $\mathcal{A}(\mathbf{A}, \mathbf{R})$ and outputs "yes" if a binary $\mathbf{K}$ exists, and "no" otherwise. If $\mathcal{A}$ succeeds with non-negligible advantage, then so does $\mathcal{D}$, contradicting the DSIS assumption. Therefore, deciding whether such a binary $\mathbf{K}$ exists is computationally hard. Moreover, finding such a $\mathbf{K}$ is also hard, as solving the search problem necessarily solves the decision problem. □

Then, we can define MatPRG as follows:

**Definition V.3** (MatPRG)**.** *Let $\mathbf{K}_1 \in \{0,1\}^{n\times l}$ be a key matrix and $\mathbf{A} \leftarrow\!\!\$\ \mathbb{Z}_q^{n\times m}$ be a randomly selected matrix. Then,* MatPRG *is defined by the following process:*

- $\mathbf{K}_2 \leftarrow \mathsf{MatRand}(1^\lambda)$*: Samples* $\mathbf{K}_2 \leftarrow\!\!\$\ \{0,1\}^{(m-n)\times l}$.
- $\mathbf{R} \leftarrow \mathsf{MatPRG}(\mathbf{A}, \mathbf{K}_1, \mathbf{K}_2)$*: Generates* $\mathbf{K} := \begin{bmatrix}\mathbf{K}_1\\ \mathbf{K}_2\end{bmatrix} \in \{0,1\}^{m\times l}$ *and outputs* $\mathbf{R} = f_{\mathbf{A}}(\mathbf{K}) \in \{0,1\}^{n\times l\times \log q}$.

**Theorem V.2.** *Let $m = n + \delta$ for some $\delta > 0$ and $\delta \cdot l > \lambda$ for a security parameter $\lambda$. Then, under the hardness of the DSIS problem over $\mathbb{Z}_q$, the output of* MatPRG *is computationally indistinguishable from uniform random. That is,* MatPRG *is a secure pseudorandom generator.*

*Proof.* Let $\mathbf{A} = [\mathbf{A}_1\ \mathbf{A}_2] \in \mathbb{Z}_q^{n\times m}$ be a uniformly sampled matrix, where $\mathbf{A}_1 \in \mathbb{Z}_q^{n\times n}$ is invertible with a high probability. Let $\mathbf{K} = \begin{bmatrix}\mathbf{K}_1\\ \mathbf{K}_2\end{bmatrix} \in \{0,1\}^{m\times l}$.

An adversary that receives $(\mathbf{A}, \mathbf{R})$, samples $\mathbf{K}_2^* \leftarrow \{0,1\}^{\delta\times l}$ at random, and computes $\mathbf{K}_1^* = \mathbf{A}_1^{-1}(\mathbf{R} - \mathbf{A}_2\mathbf{K}_2^*)$. The adversary gets $\mathbf{K}^* = \begin{bmatrix}\mathbf{K}_1^*\\ \mathbf{K}_2^*\end{bmatrix}$ and checks $\mathbf{K}_1^* \in \{0,1\}^{n\times l}$ and $\mathbf{A}\cdot\mathbf{K}^* = \mathbf{R}$. If both conditions hold, the adversary outputs "pseudorandom"; otherwise, it outputs "uniform random".

If $\mathbf{R}$ is generated as $\mathbf{A}\cdot\mathbf{K}$ for a binary $\mathbf{K} = \begin{bmatrix}\mathbf{K}_1\\ \mathbf{K}_2\end{bmatrix}$, then there exists a unique $\mathbf{K}_2 = \mathbf{K}_2^*$ for which the adversary's check will succeed. Since $\mathbf{K}_2^*$ is chosen uniformly at random, the probability is $2^{-\delta l}$.

If $\mathbf{R}$ is uniformly random, then with overwhelming probability, no such binary $\mathbf{K}$ exists, and the test fails.

Therefore, the adversary's distinguishing advantage is at most $2^{-\delta l}$, which is negligible when $\delta \cdot l > \lambda$ for a suitable security parameter $\lambda$. Therefore, under the DSIS assumption, MatPRG is a secure pseudorandom generator, as no PPT adversary can distinguish its output from uniform with non-negligible advantage. □

Let the matrix $\mathbf{A} \in \mathbb{Z}^{n\times m}$ be a globally set random matrix. In our construction, we replace the MiMC block cipher, which is originally used in CTR mode for generating pseudorandom keystreams, with a matrix-based pseudorandom generator, MatPRG. We define a symmetric key encryption scheme based on MatPRG, as in Algorithm 5, where both the input data (data) and ciphertext (ct) are matrix-formed. While MiMC is a SNARK-friendly cipher optimized for low constraint costs, our MatPRG-based approach achieves remarkable proof efficiency, requiring significantly fewer constraints per encryption.

To prove that the symmetric key encryption in Algorithm 5 is correctly performed, the prover should execute one matrix multiplication (i.e., $\mathbf{A}\cdot\mathbf{K} = \mathbf{R}$ for MatPRG) and one matrix addition (for $\mathsf{ct} \leftarrow \mathsf{data} + \mathbf{R}$). However, the former is computationally expensive, as it requires $n \cdot m \cdot l$ field multiplications. To mitigate it, we utilize Frievalds' algorithm, transforming it into proving $\mathbf{A}\cdot(\mathbf{K}\cdot\mathbf{C}) = \mathbf{R}\cdot\mathbf{C}$ where

**Algorithm 5** Symmetric key encryption scheme using MatPRG

```
Setup(1^λ) :
    $\mathbf{K}_1 \leftarrow\$ \{0,1\}^{n\times l}$
    $\mathbf{K}_2 \leftarrow \mathsf{MatRand}(1^\lambda)$
    return $(\mathbf{K}_1, \mathbf{K}_2)$
Enc($\mathbf{A}, (\mathbf{K}_1, \mathbf{K}_2), \mathsf{data} \in \mathbb{Z}^{n\times l}$)
    $\mathbf{R} \leftarrow \mathsf{MatPRG}(\mathbf{A}, \mathbf{K}_1, \mathbf{K}_2)$
    $\mathsf{ct} \leftarrow \mathsf{data} + \mathbf{R}$
    return $\mathsf{ct} \in \mathbb{Z}^{n\times l}$
Dec($\mathbf{A}, (\mathbf{K}_1, \mathbf{K}_2), \mathsf{ct} \in \mathbb{Z}^{n\times l}$) :
    $\mathbf{R} \leftarrow \mathsf{MatPRG}(\mathbf{A}, \mathbf{K}_1, \mathbf{K}_2)$
    $\mathsf{data} \leftarrow \mathsf{ct} - \mathbf{R}$
    return $\mathsf{data} \in \mathbb{Z}^{n\times l}$
```

$\mathbf{C} \leftarrow\$ \mathbb{Z}^l$. Finally, the relation for $\mathsf{RegisterData}$ using $\mathsf{MatPRG}$ is as follows:

$$R_{\mathsf{reg}} = \left\{ \begin{array}{c} \begin{pmatrix} \mathsf{ck}, \mathsf{h_k}, \mathsf{cm_{ct}}, \mathbf{A}; \\ \mathsf{ct}, \mathsf{data}, \mathbf{K}_1, \mathbf{K}_2, \mathbf{R}, \mathbf{C}, \mathsf{sk}^{\mathsf{seller}}, o \end{pmatrix} : \\ \mathbf{A} \cdot \begin{bmatrix} \mathbf{K_1} \\ \mathbf{K_2} \end{bmatrix} \cdot \mathbf{C} = \mathbf{R} \cdot \mathbf{C} \wedge \mathsf{ct} = \mathsf{data} + \mathbf{R} \wedge \\ \mathsf{cm_{ct}} = \mathsf{COM.Com}(\mathsf{ck}, \mathsf{ct}; o) \wedge \\ \mathsf{h_k} = \mathsf{CRH}(\mathsf{sk}^{\mathsf{seller}}, \mathbf{K}_1, \mathbf{K}_2) \end{array} \right\}$$

The reduced computational complexity is $m \cdot l$ field multiplications for $\mathbf{K} \cdot \mathbf{C}$, $n \cdot m$ multiplications for $\mathbf{A} \cdot (\mathbf{K} \cdot \mathbf{C})$, and $n \cdot l$ multiplications for $\mathbf{R} \cdot \mathbf{C}$. Consequently, proving $\mathsf{MatPRG}$ within the zk-SNARK circuit requires $m \cdot l + n \cdot m + n \cdot l$ multiplications. Assuming $n \approx m \approx l$, the asymptotic complexity is $\mathcal{O}(n \cdot l)$. This is effective, particularly for our registration phase. Proving the symmetric key encryption in Algorithm 2 incurs $\mathcal{O}(n \cdot l)$ hash operations, which requires 350 constraints, while the $\mathsf{MatPRG}$ approach (Algorithm 5) requires the same complexity of field multiplication, requiring only 9 constraints.

## B. Leveraging CP-SNARK

One of the significant overheads for the prover in the registration phase is checking $\mathsf{cm_{ct}}$ within the zk-SNARK circuit, as COM.Com computation is proportional to the data size. Namely, encoding the COM.Com within the circuit imposes a substantial amount of proving time, particularly for large-sized data. To reduce the proving time required for checking $\mathsf{cm_{ct}}$, we employ CP-SNARK, which allows a commitment to the witness to be provided as input along with the proof. Consequently, proving $\mathsf{cm_{ct}}$ within the circuit is smoothly replaced by committing to ct using CP-SNARK. Therefore, a seller can take the commitment operation out of the circuit, leading to reduced proving overhead.

## VI. Evaluation

### A. Implementation

We implement zkMarket using Arkworks [32] and smart contracts deployed on the Ethereum Hardhat [33] test network. We instantiate both zk-SNARK and CP-SNARK using LegoSNARK [24], based on the Groth16 proving system (CP-Groth16). Although Groth16 requires a trusted setup, it offers minimal on-chain gas cost, which we consider a favorable trade-off for blockchain applications. For the public key encryption scheme, we employ the ElGamal encryption scheme. For the collision-resistant hash function, we use the MiMC7 SNARK-friendly hash function [22]. All the benchmarks are evaluated on the Macbook M1 Pro with 32GB of RAM.

### B. Performance analysis

*1) Performance analysis on the registration phase*

Figure 3(a) illustrates the proving time performance for the registration phase, where we compare our optimized zkMarket with SmartZKCP [19] and VECK by Tas et al. [10]. Note that some larger data sizes and configurations couldn't be tested due to hardware limits.

zkMarket, leveraging $\mathsf{MatPRG}$ and CP-Groth16, achieves significant efficiency gains. A baseline configuration, similar to prior works like SmartZKCP using MiMC hash functions with standard Groth16, requires approximately 7.4 seconds (747,156 constraints) to prove 32KB of data. Our initial optimization with CP-Groth16 reduces this to 4.04 seconds (374,783 constraints). Crucially, by integrating our novel

**Algorithm 6** Register phase with MatPRG

```
Off − chain
RegisterData($1^\lambda$, crs, $\mathbf{A}$, data, $\mathsf{sk}^{\mathsf{seller}}$) :
    $(\mathbf{K}_1, \mathbf{K}_2) \leftarrow \mathsf{SE.Gen}(1^\lambda)$
    $\mathsf{ct} \leftarrow \mathsf{SE.Enc}(\mathbf{A}, (\mathbf{K}_1, \mathbf{K}_2), \mathsf{data})$
    $\mathsf{h_k} \leftarrow \mathsf{CRH}(\mathsf{sk}^{\mathsf{seller}} || \mathbf{K}_1 || \mathbf{K}_2)$
    $o \leftarrow\$ \mathbb{Z}$
    $\mathsf{cm_{ct}} \leftarrow \mathsf{COM.Com}(\mathsf{ck}, \mathsf{ct}; o)$
    $\mathbf{C} \leftarrow\$ \mathbb{Z}^l$
    $\mathbf{x} := (\mathsf{h_k}, \mathbf{A}, \mathsf{ck}, \mathsf{cm_{ct}})$
    $\mathbf{w} := (\mathsf{ct}, \mathsf{data}, \mathbf{K}_1, \mathbf{K}_2, \mathbf{R}, \mathbf{C}, \mathsf{sk}^{\mathsf{seller}}, o)$
    $\pi_{reg} \leftarrow \Pi_{\mathsf{snark}}.\mathsf{Prove}(\mathsf{crs}_{reg}, \mathsf{ck}, \mathbf{x}; \mathbf{w})$
    return $\mathsf{tx}_{reg} = (\mathsf{h_k}, \mathsf{cm_{ct}}, \pi_{reg})$

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

TABLE II: Evaluation of GenerateTrade and AcceptTrade with 32 depth Merkle tree

| Algorithm | Constraints | CRS (MB) | Setup (s) | Prove (s) | Verify (ms) | Gas |
|---|---|---|---|---|---|---|
| GenerateTrade | 12,882 | 4.3 | 0.19 | 0.2 | 1.6 | 1,996,915 |
| AcceptTrade | 24,210 | 9.3 | 0.2 | 0.38 | 1.6 | 1,378,750 |

MatPRG for efficient encryption proof alongside CP-SNARK, the total time for proving 32KB data dramatically drops to just 0.191 seconds (16,575 constraints). Through these comprehensive optimizations, zkMarket demonstrates at least five times faster performance than SmartZKCP for data sizes ranging from 16KB to 64KB.

For larger datasets, zkMarket also significantly outperforms VECK. Experimental results show that generating a zk-SNARK proof for 1,024 KB (1 MB) of data takes only 2.89 seconds with zkMarket, whereas VECK (using ElGamal) requires 703.6 seconds for the identical operation. This highlights zkMarket's unparalleled efficiency in handling large-scale data, achieving roughly a 367x speedup over an unoptimized baseline (e.g., 1,030 seconds for 1MB) and approximately 250 times faster than VECK.

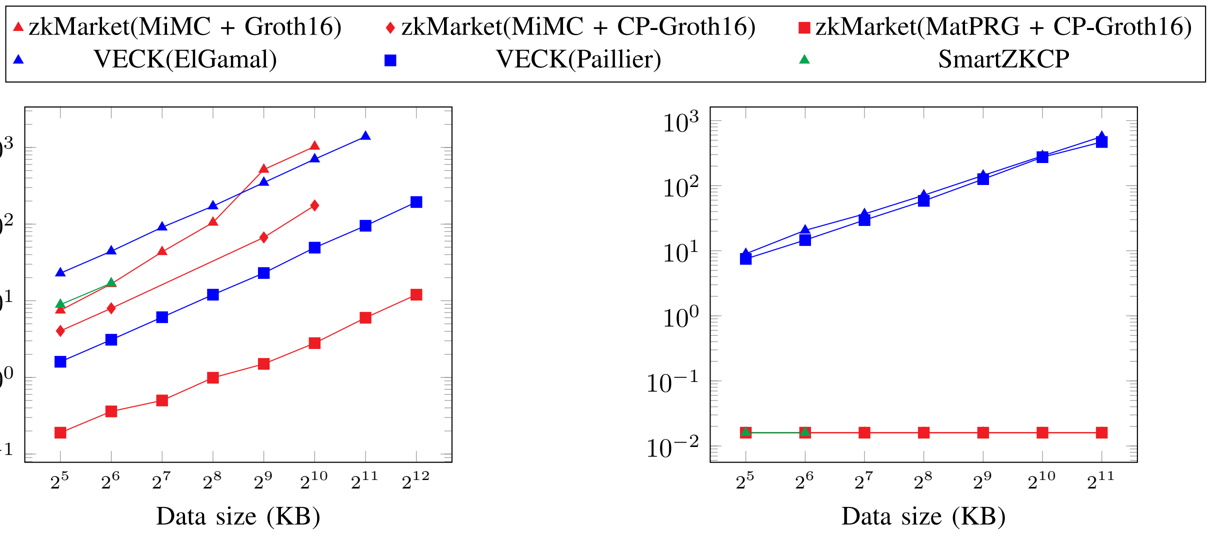
Figure 3(b) illustrates verification time for data registration. The on-chain cost of the SC.RegisterData function working on the smart contract only requires 303,415 gas. Both zkMarket and SmartZKCP utilize Groth16 zk-SNARK, offering a constant verification cost of just 1.6 milliseconds. In contrast, VECK's verification time increases proportionally to the size of the data, further emphasizing zkMarket's scalability.

### 2) *Performance evaluation of the trade generation and acceptance phases*

Table II shows the performance of the GenerateTrade and AcceptTrade algorithms with a Merkle tree of depth 32. Unlike the RegisterData operation, GenerateTrade and AcceptTrade are executed each time a transaction occurs. However, a key characteristic is that GenerateTrade and AcceptTrade are not impacted by the size of the data being traded, whereas RegisterData is. Specifically, GenerateTrade proves the buyer's ability to pay, while AcceptTrade proves the correctness of the decryption key. Both operations involve generating zk-SNARK proofs for a fixed-size fee and key, allowing GenerateTrade and AcceptTrade to run in constant time regardless of the data size. The proving time and the verification time for the GenerateTrade operation are approximately 0.2s and 1.6ms, respectively. Similarly, for the AcceptTrade operation, the proving time is around 0.38s and the verification time is approximately 1.6ms.

## Acknowledgment

This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIP) (No.2021-0-00518, Blockchain privacy preserving techniques based on data encryption, 50%, No.2021-0-00532, Blockchain scalability solutions supporting high performance/capacity transactions, 30%, No.2021-0-00528, No.2021-0-00727, A Study on Cryptographic Primitives for SNARK, 20%). Jihye Kim and Hyunok Oh are co-corresponding authors.



(a) Proving time

(b) Verification time

Fig. 3: Comparison of performance

## References

[1] H. Pagnia, F. C. Gärtner *et al.*, "On the impossibility of fair exchange without a trusted third party," Citeseer, Tech. Rep., 1999.

[2] T. Jung, X.-Y. Li, W. Huang, J. Qian, L. Chen, J. Han, J. Hou, and C. Su, "Accounttrade: Accountable protocols for big data trading against dishonest consumers," in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.

[3] W. Dai, C. Dai, K.-K. R. Choo, C. Cui, D. Zou, and H. Jin, "Sdte: A secure blockchain-based data trading ecosystem," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 725–737, 2019.

[4] B. Wang, B. Li, Y. Yuan, C. Dai, Y. Wu, and W. Zheng, "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)*. IEEE, 2022, pp. 968–975.

[5] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[6] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[7] Y. Zhao, Y. Yu, Y. Li, G. Han, and X. Du, "Machine learning based privacy-preserving fair data trading in big data market," *Information Sciences*, vol. 478, pp. 449–460, 2019.

[8] S. Dziembowski, L. Eckey, and S. Faust, "Fairswap: How to fairly exchange digital goods," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 967–984.

[9] D. Sheng, M. Xiao, A. Liu, X. Zou, B. An, and S. Zhang, "Cpchain: a copyright-preserving crowdsourcing data trading framework based on blockchain," in *2020 29th international conference on computer communications and networks (ICCCN)*. IEEE, 2020, pp. 1–9.

[10] E. N. Tas, I. A. Seres, Y. Zhang, M. Melczer, M. Kelkar, J. Bonneau, and V. Nikolaenko, "Atomic and fair data exchange via blockchain," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 3227–3241.

[11] G. Su, W. Yang, Z. Luo, Y. Zhang, Z. Bai, and Y. Zhu, "Bdtf: A blockchain-based data trading framework with trusted execution environment," in *2020 16th International Conference on Mobility, Sensing and Networking (MSN)*. IEEE, 2020, pp. 92–97.

[12] C. Chenli, W. Tang, and T. Jung, "Fairtrade: Efficient atomic exchange-based fair exchange protocol for digital data trading," in *2021 IEEE International Conference on Blockchain (Blockchain)*. IEEE, 2021, pp. 38–46.

[13] C. Chenli, W. Tang, H. Lee, and T. Jung, "Fair 2 trade: Digital trading platform ensuring exchange and distribution fairness," *IEEE Transactions on Dependable and Secure Computing*, 2024.

[14] B. Wiki, "Hash time locked contracts," https://en.bitcoin.it/wiki/Hash_Time_Locked_Contracts, accessed: 2024-06-17.

[15] R. Gennaro, C. Gentry, B. Parno, and M. Raykova, "Quadratic span programs and succinct nizks without pcps," in *Advances in Cryptology–EUROCRYPT 2013: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings 32*. Springer, 2013, pp. 626–645.

[16] J. Groth, "On the size of pairing-based non-interactive arguments," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2016, pp. 305–326.

[17] A. Gabizon, Z. J. Williamson, and O. Ciobotaru, "Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge," *Cryptology ePrint Archive*, 2019.

[18] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *2018 IEEE symposium on security and privacy (SP)*. IEEE, 2018, pp. 315–334.

[19] X. Liu, J. Zhang, Y. Wang, X. Yang, and X. Yang, "Smartzkcp: Towards practical data exchange marketplace against active attacks," *Cryptology ePrint Archive*, 2024.

[20] S. Avizheh, P. Haffey, and R. Safavi-Naini, "Privacy-preserving fairswap: Fairness and privacy interplay," *Proceedings on Privacy Enhancing Technologies*, 2022.

[21] G. Jeong, N. Lee, J. Kim, and H. Oh, "Azeroth: Auditable zero-knowledge transactions in smart contracts," *IEEE Access*, vol. 11, pp. 56 463–56 480, 2023.

[22] M. R. Albrecht, L. Grassi, C. Rechberger, A. Roy, and T. Tiessen, "Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity," in *ASIACRYPT*, 2016, pp. 191–219.

[23] L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, and M. Schofnegger, "Poseidon: A new hash function for {Zero-Knowledge} proof systems," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 519–535.

[24] M. Campanelli, D. Fiore, and A. Querol, "Legosnark: Modular design and composition of succinct zero-knowledge proofs," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 2075–2092.

[25] B. Wiki, "Zero knowledge contingent payment," https://en.bitcoin.it/wiki/Zero_Knowledge_Contingent_Payment, accessed: 2024-06-17.

[26] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," *Communications of the ACM*, vol. 59, no. 2, pp. 103–112, 2016.

[27] Y. Li, C. Ye, Y. Hu, I. Morpheus, Y. Guo, C. Zhang, Y. Zhang, Z. Sun, Y. Lu, and H. Wang, "Zkcplus: Optimized fair-exchange protocol supporting practical and flexible data exchange," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 3002–3021.

[28] M. Bellare, A. Boldyreva, A. Desai, and D. Pointcheval, "Key-privacy in public-key encryption," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2001, pp. 566–582.

[29] E. Ben Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *2014 IEEE Symposium on Security and Privacy*, 2014, pp. 459–474.

[30] Z. Guan, Z. Wan, Y. Yang, Y. Zhou, and B. Huang, "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. [Online]. Available: https://eprint.iacr.org/2019/1354

[31] V. Lyubashevsky, "Lattice signatures without trapdoors," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2012, pp. 738–755.

[32] arkworks contributors, "`arkworks` zksnark ecosystem," 2022. [Online]. Available: https://arkworks.rs

[33] "Hardhat." [Online]. Available: https://hardhat.org

## Appendix A
## Formal definitions

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

The symmetric encryption scheme $\mathsf{SE}$ ensures indistinguishability under chosen-plaintext attack (IND-CPA) security and key indistinguishability under chosen-plaintext attack (IK-CPA [28]) security.

### B. 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.

- $\mathsf{Gen}(1^\lambda) \rightarrow (\mathsf{sk}, \mathsf{pk})$ : returns a key pair $(\mathsf{sk}, \mathsf{pk})$ for secret key $\mathsf{sk}$ and public key $\mathsf{pk}$ taking a security parameter $1^\lambda$ 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 [28]) security.

### C. 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 witnesses $\mathbf{w}$ from a proof $\pi$ which passes the verification. For the *succinctness*, it means that the proof size and the verification time are logarithmic in the size of the witness. A SNARK may satisfy *zero-knowledge* when nothing about the witness is leaked from the proof. We refer to such SNARK as zk-SNARK, and it can be constructed with the simulator which outputs a valid proof without knowing the witness $\mathbf{w}$.

### D. Commit-and-Prove SNARK

A CP-SNARK $\Pi_{\mathsf{cp}}$ = (Setup, Prove, Verify, VerifyCom) works as follows:

- $\mathsf{Setup}(R) \to (\mathsf{ck}, \mathsf{ek}, \mathsf{vk})$: 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.
- $\mathsf{Prove}(\mathsf{ek}, \mathbf{x}, \mathbf{w}) \to (\pi, c; o)$: 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 $\pi$, a commitment $c$, and an opening $o$ such that $\mathsf{VerifyCom}(\mathsf{ck}, c, u, o) = 1$.
- $\mathsf{Verify}(\mathsf{vk}, \mathbf{x}, \pi, c) \to 0/1$ : takes a verification key vk, a statement $\mathbf{x}$, a commitment $c$, and a proof $\pi$ as inputs, and outputs 1 if $\mathbf{x}, c, \pi$ is within the relation $\mathcal{R}$, or 0 otherwise.
- $\mathsf{VerifyCom}(\mathsf{ck}, c, u, o) \to 0/1$ : 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 A.1.** *CP-SNARK satisfies completeness, succinctness, knowledge soundness, zero-knowledge, and binding.*

### E. Pseudorandom Generator

**Definition A.2** (PRG, Pseudorandom Generator). *Let function* $G : \{0,1\}^n \to \{0,1\}^m$ *with* $m > n$ *is pseudorandom generator. Then, for all PPT adversaries* $\mathcal{A}$ *and random* $y \in \{0,1\}^m$ *and pseudorandom* $G(x)$ *for a random seed* $x \in \{0,1\}^n$*, there is a negligible function* negl *such that*

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

## APPENDIX B
## RELATIONS FOR ZKMARKET

As mentioned previously in Section IV-B, there exist three relations for each phase of zkMarket. Here we provide details of the respective relations.

### A. Relation for the registration phase

Formally, the relation for the registration phase $R_{\mathsf{reg}}$ is as follows:

$$R_{\mathsf{reg}} = \left\{ \begin{array}{c} \left(\mathsf{ck}, \mathsf{h_k}, \mathsf{cm_{ct}}; \mathsf{ct}, \mathsf{data}, \mathsf{k}, \mathsf{sk}^{\mathsf{seller}}, o\right) : \\ \mathsf{h_k} = \mathsf{CRH}(\mathsf{sk}^{\mathsf{seller}}, \mathsf{k}) \land \mathsf{ct} = \mathsf{SE.Enc}(\mathsf{k}, \mathsf{data}) \land \\ \mathsf{cm_{ct}} = \mathsf{COM.Com}(\mathsf{ck}, \mathsf{ct}; o) \end{array} \right\}$$

### B. Relation for the trade generation phase

The relation $R_{\mathsf{gen}}$ is as follows:

$$R_{\mathsf{gen}} = \left\{ \begin{array}{c} \begin{pmatrix} \mathsf{cm_{order}}, \mathsf{ct_{order}}, \mathsf{sct_{new}}, \mathsf{sct_{old}}; \\ r, \mathsf{h_k}, \mathsf{pk}^{\mathsf{seller}}, \mathsf{pk}^{\mathsf{buyer}}, \mathsf{k}^{\mathsf{ENA}}, \mathsf{fee} \end{pmatrix} : \\ \mathsf{cm_{order}} = \mathsf{COM.Com}(\mathsf{ck}, \mathsf{pk}^{\mathsf{seller}}, \mathsf{fee}, \mathsf{h_k}, \mathsf{pk}^{\mathsf{buyer}}; r) \land \\ \mathsf{ct_{order}} = \mathsf{PKE.Enc}(\mathsf{pk}^{\mathsf{seller}}, \mathsf{fee}, \mathsf{h_k}, \mathsf{pk}^{\mathsf{buyer}}, r) \land \\ \mathsf{fee} = \mathsf{SE.Dec}(\mathsf{k}^{\mathsf{ENA}}, \mathsf{sct_{old}}) - \mathsf{SE.Dec}(\mathsf{k}^{\mathsf{ENA}}, \mathsf{sct_{new}}) \end{array} \right\}$$

### C. Relation for the trade acceptance phase

The relation $R_{\mathsf{acc}}$ is defined as:

$$R_{\mathsf{acc}} = \left\{ \begin{array}{c} \begin{pmatrix} \mathsf{rt}, \mathsf{nf}, \mathsf{cm_{Azeroth}}, \mathsf{h_k}, \mathsf{ct_k}, \mathsf{pk}^{\mathsf{seller}}, \mathsf{addr}^{\mathsf{seller}}; \\ \mathsf{cm_{order}}, \mathsf{Path}, \mathsf{sk}^{\mathsf{seller}}, \mathsf{k}, \mathsf{pk}^{\mathsf{buyer}}, r, \mathsf{fee}, \mathsf{o_{Azeroth}} \end{pmatrix} : \\ \mathsf{ct_k} = \mathsf{PKE.Enc}(\mathsf{pk}^{\mathsf{buyer}}, \mathsf{k}) \land \\ \mathsf{h_k} = \mathsf{CRH}(\mathsf{sk}^{\mathsf{seller}}, \mathsf{k}) \land \mathsf{nf} = \mathsf{CRH}(\mathsf{cm_{order}}, \mathsf{sk}) \land \\ \mathsf{cm_{order}} = \mathsf{COM.Com}(\mathsf{ck}, \mathsf{pk}^{\mathsf{seller}}, \mathsf{fee}, \mathsf{h_k}, \mathsf{pk}^{\mathsf{buyer}}; r) \land \\ \mathsf{cm_{Azeroth}} = \mathsf{COM.Com}(\mathsf{ck}, \mathsf{fee}, \mathsf{addr}^{\mathsf{seller}}; \mathsf{o_{Azeroth}}) \land \\ \mathsf{MT.MemVerify}(\mathsf{rt}, \mathsf{cm_{order}}, \mathsf{Path}) = 1 \end{array} \right\}$$