Privatizing the Volume and Timing of Blockchain Transactions

Trevor J. Miller

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Science & Applications

Thang Hoang, Chair
Lenwood S. Heath
Peng Gao

February 10, 2023
Blacksburg, Virginia

Keywords: blockchain, privacy, zero-knowledge, timestamp

Privatizing the Volume and Timing of Blockchain Transactions

Trevor J. Miller

(ABSTRACT)

With current state-of-the-art privacy-preserving blockchain solutions, users can submit transactions to a blockchain while maintaining full anonymity and not leaking the contents of the transaction through cryptographic techniques like zero-knowledge proofs and homomorphic encryption. However, the architecture of a blockchain consists of a decentralized network where every network participant maintains their own local copy of the blockchain and updates it upon every added transaction. As a result, the volume of blockchain transactions and the timestamp of each blockchain transaction for an application is publicly available. This is problematic for applications with time-sensitive or volume-sensitive outcomes because users may want this information to be privatized, such as not leaking the lateness of student examinations. However, this is not possible with existing blockchain research. In this thesis, we propose a blockchain system for multi-party applications that does not leak any useful information from the volume and timing metadata of the application's transactions, including maintaining the privacy of a time-sensitive or volume-sensitive outcome. We achieve this by adding sufficient noise using indistinguishable decoy transactions such that an adversary cannot deduce which transactions actually impacted the outcome of the application. This is facilitated in a manner where anyone can publicly verify the application's execution to be correct, fair, and honest. We demonstrate and evaluate our approach by implementing a Dutch auction that supports decoy bid transactions on a private Ethereum blockchain network.

Privatizing the Volume and Timing of Blockchain Transactions

Trevor J. Miller

(GENERAL AUDIENCE ABSTRACT)

Blockchains are distributed, append-only, digital ledgers whose current state is continuously agreed upon through the consensus of network participants and not by any centralized party. These characteristics make them unique for many applications because they enable the application to be facilitated and executed in a public, verifiable, decentralized, and tamper-proof manner. For example, Bitcoin, the most popular cryptocurrency, uses blockchains to continuously maintain a permanent, verifiable ledger of payment transactions. However, one downside of this public architecture is that the volume of transactions and the timestamp of each transaction can always be publicly observed (e.g. the timestamp of every Bitcoin payment is public). This is problematic for applications with time-sensitive or volume-sensitive outcomes because users may want this volume and timing information to be privatized, such as not leaking the lateness of student examinations which could have severe consequences like violating student privacy laws. But currently with state-of-the-art blockchain research, privatizing this information is not possible. In this thesis, we demonstrate our approach that enables these time-sensitive and volume-sensitive applications to be implemented on blockchains in a manner that can maintain the privacy of these time-sensitive or volume-sensitive outcomes without sacrificing the application's integrity or verifiability. We then demonstrate and evaluate our approach through implementing a Dutch auction that supports decoy bid transactions on a private blockchain network.

Contents

List of Figures vi

List of Tables vii

1 Introduction 1

1.1 Contributions 4

1.2 Problem Overview 5

1.2.1 What Must Be Public in Blockchain Frameworks? 5

1.2.2 Timing and Volume Analysis 7

2 Related Work 11

2.1 Privacy Preserving Blockchain Applications 11

2.2 Timing Metadata on Blockchains 12

2.3 Privatizing Which Computation Was Performed 12

2.4 Mitigating Through High Frequency 13

3 Proposed Method 14

3.1 Preliminaries 14

3.1.1 Notation 14

3.2 Models . . . 16
3.2.1 System Model . . . 16
3.2.2 Threat Model . . . 18
3.2.3 Security Guarantees . . . 19
3.3 Proposed Method . . . 19
3.3.1 Overview . . . 19
3.3.2 The Application's Target Function . . . 21
3.3.3 Detailed Algorithm . . . 24
3.3.4 Security Analysis . . . 28

4 Experiment **31**
4.1 Implementation . . . 31
4.2 Hardware . . . 33
4.3 Parameters . . . 33
4.4 Metrics . . . 34
4.5 Results . . . 34
4.5.1 Generic Application . . . 34
4.5.2 Dutch Auction . . . 37

5 Conclusion **40**

Bibliography **41**

List of Figures

1.1 Leaking student grades for a class of 5 students with a submission deadline of 12:00 AM. . . . 3

1.2 Leaking a seller's revenue in a Dutch auction to be $71. . . . 3

1.3 Exploiting volume and timing information in a Dutch auction to obtain a lower price. . . . 4

1.4 Example ledger for a privacy-preserving blockchain where transactions can have three functionalities: mint, send, and lend. . . . 7

3.1 System model figure . . . 18

3.2 Example distribution of real versus decoy transactions for an application. . . . 20

3.3 An adversary's view of the distribution from Figure 3.2. . . . 20

3.4 A deadline-based application using our approach with K=2 time periods. . . . 23

3.5 An adversary's view of the distribution in Figure 3.4. . . . 24

4.1 Architecture diagram for our implementation. . . . 33

4.2 Average end-to-end time delay (proof generation + blockchain confirmation) per bidder in a Dutch auction. . . . 38

4.3 End-to-end time delay (proof generation + blockchain confirmation) for an auctioneer in a Dutch auction. . . . 39

List of Tables

4.1 Local overhead for our zkSNARK circuits (i.e. proof and key generation). . . 34

4.2 Blockchain overhead for our zkSNARK circuits (i.e. verification and library generation). 35

4.3 Blockchain overhead for our proposed method with N Inputs (N = Real + Decoy). 35

Chapter 1

Introduction

Recently, blockchains have gained popularity and have begun to go mainstream through cryptocurrencies like Bitcoin. Many applications and sectors have been touted to be revolutionized by blockchains due to the unique characteristics that they offer [1] [2]. Specifically, a blockchain is a public, decentralized, distributed, and append-only ledger of transactions whose current state is continuously agreed upon through the consensus of network participants and not by any centralized party [3]. For example, the financial sector may be revolutionized by the use of blockchains such as Bitcoin that maintain a permanent, decentralized, and verifiable ledger of digital payment transactions [4].

For blockchains to realize their potential though, they must be able to be support applications that have privacy requirements. But due to their decentralized architecture, blockchains are prone to leak privacy. For example, Bitcoin reveals every payment amount, every user's balance (through pseudonyms), and every user's transaction history [4]. This is publicly available because, as explained, the ledger of transactions is stored and updated in a decentralized manner [5]. To address these privacy concerns, privacy-preserving blockchain solutions have been developed. Existing privacy-preserving blockchains have successfully shown how to achieve full anonymity and confidentiality of a transaction's contents while maintaining the integrity and verifiability of the system [6] [7] [8] [9]. For example, the ZCash blockchain is able to support fully anonymous, private cryptocurrency payments [7].

However, all privacy-preserving blockchain solutions still leak volume and timing metadata

for every transaction. This is because each transaction still needs to be added to the blockchain at some point, and since the blockchain is decentralized, anyone can observe when and how many transactions are added. In many applications, this volume and timing metadata being leaked can bring about privacy issues and affect the fairness of outcomes, especially when combined with auxiliary information. This is often the case when an application has an outcome that is time-sensitive or volume-sensitive.

One example of a time-sensitive application is grading student examinations. Blockchains have been touted to be used for grading and storing student examinations because they would enable these examinations to a) be graded verifiably and trustlessly without any centralized party and b) be stored in a tamper-proof and verifiable manner [10]. But in an educational context, student privacy is often enforced by laws and regulations, so, in practice, blockchain based grading systems must be able to achieve privacy of students' exam submissions and their grades. If a grading policy that includes penalties for late submissions (time-dependent) and resubmissions (volume-dependent) is implemented on a blockchain using existing privacy-preserving techniques, each student could submit their exam verifiably while achieving anonymity and maintaining the confidentiality of their submission(s). However, the public volume and timing metadata of these submission transactions will leak the timestamps of all submissions (i.e. if a submission was late or not) and the total number of submissions. This can then be used to infer the students' grades, which is information that should be kept confidential in order to satisfy student privacy laws. To demonstrate, assume a class of five students, and exams are penalized 10% for being late after a public deadline (e.g. 12:00 AM) and 10% for each time the exam is resubmitted. Anyone could then observe, for example, that there were ten submissions that were all late which would reveal a maximum class average of 80%. This can be viewed in Figure 1.1.

Another example is a Dutch auction. A Dutch auction is a type of auction where the seller

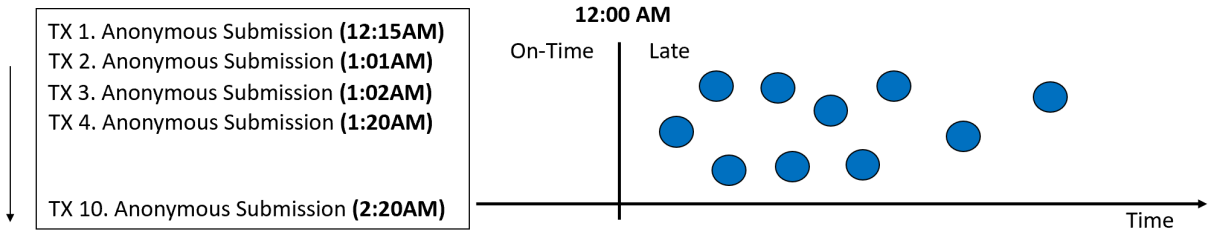

Figure 1.1: Leaking student grades for a class of 5 students with a submission deadline of 12:00 AM.

sells one or more items by initially listing at a high asking price and incrementally lowering the price over time until all items have been sold [11]. The price decrements are typically done at publicly defined times (e.g. the items cost \$10 on day one, \$5 on day two, and so on). In other words, buyers will pay a publicly known price for items dependent on their time of purchase. Because of the properties of such an auction, if the timing and volume of sales are public (which will be the case even when using privacy-preserving blockchains), this will leak certain information that the seller may want to keep confidential. Firstly, one can calculate the seller's revenue using the public prices and the volume and timing metadata of each sale transaction. This can be viewed in Figure 1.2 where the seller's revenue can be calculated to be \$71. Secondly, buyers can exploit this information to obtain a lower, more optimal buying price. For example, a bidder can see there have been no purchases yet (volume), and thus, they know they can wait until the price decreases due to the little risk of the items selling out. This can be viewed in Figure 1.3 where the buyer can exploit this information to wait until day three to only pay \$1.

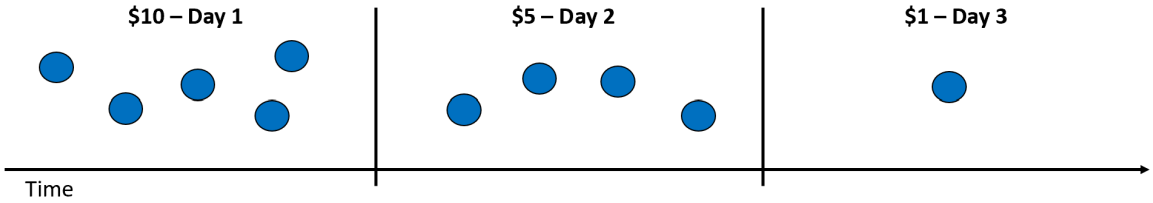

Figure 1.2: Leaking a seller's revenue in a Dutch auction to be \$71.

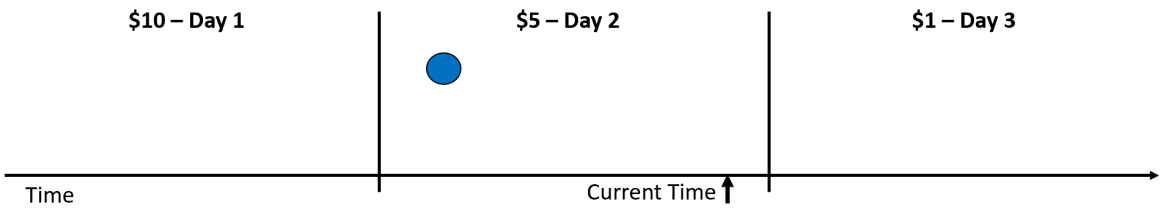

Figure 1.3: Exploiting volume and timing information in a Dutch auction to obtain a lower price.

Thus, we ask the following question: Given that the timing and volume of transactions for applications on blockchains leaks significant information, can we design a solution for such applications that does not leak volume and timing information while not sacrificing the integrity or functionality of the application?

1.1 Contributions

We answer this question by proposing a new system to implement multi-party privacy-preserving blockchain applications. In such applications, multiple parties want to execute some joint computation (i.e. the application's logic) using secret inputs in a manner that does not leak their inputs to others. In addition to maintaining the privacy of parties' inputs for these applications, our system is able to maintain volume and timing privacy such that the volume and timing metadata of an application's blockchain transactions will leak no useful information, even after the application has been finalized. This makes our approach ideal for applications with time-sensitive or volume-sensitive outcomes (e.g. a student exam or Dutch auction). We achieve this in a manner where anyone can publicly verify the correctness and fairness of the application's execution. To our knowledge, we are the first to propose a solution that achieves volume and timing privacy for blockchain applications.

1.2 Problem Overview

Blockchains are a distributed, tamper-proof, decentralized data storage solution. They offer a different set of characteristics than other data storage solutions such as client-server databases, and, as a result, they have different architectures. Specifically, blockchains are stored in a distributed manner where any network participant can view the contents of the blockchain at any time. As a result, certain information may be publicly known in blockchain systems that may be privatized in other systems. For example, the timestamp that some storage was updated on the blockchain will always be publicly known to everyone because all contents are stored and updated in a distributed manner. In a typical client-server database model, these timestamps can easily be kept private between the client and server and not leaked publicly to everyone.

When information like timestamps is public to everyone in blockchain settings, this may leak information or bring about privacy issues for certain blockchain applications. For example, if everyone can observe that some student submitted their exam late via its public blockchain timestamp, this may leak information about their grade and violate student privacy laws [12]. In spite of the blockchain's public nature, existing privacy-preserving blockchain solutions have shown how to successfully achieve full anonymity and privatize the contents included within blockchain transactions. However, there is certain metadata like timestamps that can not be privatized due to the core architecture of decentralized blockchains and how they are distributed.

1.2.1 What Must Be Public in Blockchain Frameworks?

In this thesis, we make the following observations about the architecture of all blockchains (standard and privacy-preserving):

- All blockchains can be abstracted to a sequence of public state transitions, $S_0 \rightarrow S_1 \rightarrow \cdots \rightarrow S_N$ where S_0 is the genesis state and S_N is the current state.
- To go from any S_i to S_{i+1}, all validating nodes must be able to verify it is a valid state transition by executing the same publicly known verification logic, $\{0, 1\} \leftarrow \text{Verify}(S_i, S_{i+1})$ where 1 is valid and 0 is invalid.
- State transitions for different applications implemented on the same blockchain will have different verifying logic (e.g. VerifyAuction(S_i, S_{i+1}) vs. VerifyGrades(S_i, S_{i+1}) vs. VerifyPayment(S_i, S_{i+1}))). An example is verifying that a payment does not cause a user to have a negative balance. Even for privacy-preserving computations, there needs to be some public logic to make the computation verifiable (oftentimes, in the form of verifying a cryptographic proof). This is to maintain the integrity of the system, but, as a result, the volume of transactions for an application is leaked.
- Due to all state transitions being public, there will be some public metadata that corresponds to each one that cannot be privatized. Such metadata includes a) the timestamp of S_i, b) how many times previous state transitions with the same verifying logic was executed, and c) current state transition number (e.g. S_5 versus S_{123}). In this thesis, we will focus mainly on a) and b), but note that any public metadata can be substituted wherever timestamps are mentioned.

Using our observations from above, Figure 1.4 is an example set of transactions that demonstrates that the volume and timing of transactions for every application is always leaked, even on privacy-preserving blockchains.

Throughout this thesis, we recognize that this volume and timing information being public is core to how blockchains operate. To our knowledge, there has been no proposed blockchain architecture which can hide any of this information, and it is possible that there may never

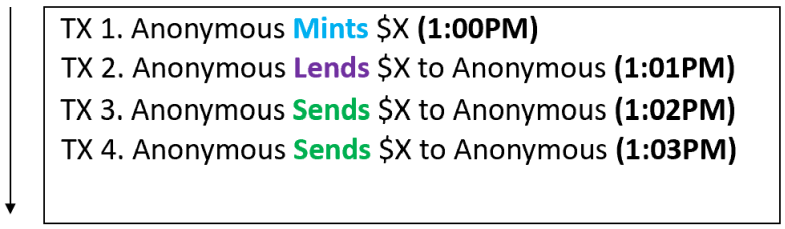

Figure 1.4: Example ledger for a privacy-preserving blockchain where transactions can have three functionalities: mint, send, and lend.

will be. As a result, we do not attempt to propose any changes to the underlying blockchain's architecture.

1.2.2 Timing and Volume Analysis

In particular, this thesis focuses on two bits of information leakages for a blockchain application: the timing of its transactions and the volume of its transactions. As demonstrated in the previous section, both these are leaked due to a blockchain's decentralized architecture. This is currently true on all blockchains: standard and privacy-preserving.

When this information is public, it leaves the set of transactions for an application open to volume and timing analysis, which could leak information to adversaries. This can have serious consequences such as violating privacy laws, affecting the fairness of an application's outcome, or leaking other auxiliary information. In this section, we explore how volume and timing analysis can be used maliciously in different types of applications which will demonstrate a need to develop a solution to privatize this information.

Exploiting Pending Transactions

In multi-party applications, parties can potentially use volume and timing analysis to manipulate the application and gain a more favorable outcome for themselves. Applications are

especially prone to be manipulated if this information can be learned while the transactions are still pending (i.e. consensus has not been reached yet).

For a concrete example, one Ethereum (the most popular blockchain for applications) user resorted to bribing a block producer over $10,000 (5 ETH) to generate a fully private block consisting of only their transactions in order to execute a large-scale purchase of a hundred valuable, digital collectibles (i.e. CryptoPunks) [13] [14] [15] . These collectibles were all listed for sale with public prices on a marketplace by their respective owners. This can be viewed in Ethereum block #12929843. Until officially confirmed and posted on the blockchain, all transactions were kept in a completely private, local pending queue by this block producer so that no one else could ever view these transactions while they were still pending. The buyer chose this method because there was fear that if a high volume of pending purchase transactions could be observed at one time, bots may attempt to exploit this by front-running these purchases, and this would result in failed purchases or less optimal prices for the original buyer. Even if existing privacy-preserving blockchain solutions were used, the marketplace prices still need to be public, so the problem still persists.

In fact, there is a whole industry of these bots called maximal extractable value (MEV) bots which observe the public volume and timing metadata of an applications' pending transactions and reorder the transactions within a block in a way that is most favorable or profitable for them. It is estimated that these MEV bots have profited $674 million from users since 2020 on the Ethereum blockchain [16] [17].

Analyzing Posted Transactions

Even if transactions can get past the pending queue unobserved, multi-party applications are still prone to be manipulated through observing the timing and volume of posted trans-

actions. For example, in existing sealed-bid auction implementations on blockchains (i.e. standard auctions which maintain confidentiality of bid amounts), even if bid transactions can be made fully anonymously and maintain the privacy of the bid amount, each bid must still be submitted through a blockchain transaction [6] [18] [19] [20] [21] [22]. Bidders then may be able to analyze the timing and volume of other bid transactions in order to learn information, gain competitive advantages, and potentially manipulate the outcome. For example, auction theory states that the more bids there are, the higher probability the winning bid will be higher. This indirectly leaks information such as the amount of competition that bidders can exploit (e.g. a bidder who can observe that there have been no previous bids knows that a small $1 bid will win). It has also been shown that auctions that hide the volume of bids on average attract more bidders due to greater competition uncertainty [23] [24] [25]. As a result, the volume of bids is information that a seller may want to privatize.

Time-Based Applications

Applications where the execution time of its transactions directly affects the outcome are very common on blockchains because the blockchain can be used as a universally trusted timekeeper. In fact, we analyzed the logic of 4,238 applications on Ethereum (the most popular blockchain for applications), and we found that 2,893 out of 4,238 or 68.263% directly used a concept of time in their logic (denoted by the *block.timestamp* or *block.number* keywords in the application's code). The source code of these applications were verified and provided by Etherscan on June 11, 2022 at 1:10 PM EST [26]. With existing research, there is no way to privately but verifiably use the execution timestamp (or block number) within an application's code because timing metadata is always public. Thus, these time-sensitive applications can not currently be implemented on blockchains in a privacy-preserving manner, even with state-of-the-art solutions.

How Do Existing Applications Protect Against Volume and Timing Analysis?

Blockchain applications currently are forced to accept volume and timing information will be public and design their application with this in mind. If this information being public is not acceptable for an application, the only option they have is to not use a blockchain altogether. Thus, we believe there is a clear need to demonstrate a solution to privatize such volume and timing information.

Chapter 2

Related Work

2.1 Privacy Preserving Blockchain Applications

Existing solutions for privacy-preserving blockchain applications can be categorized into three categories: minimally trusted manager, secure multi-party computation (sMPC), and transaction-based. We believe our work is complementary to all three categories such that each of these categories can be extended to support volume and timing privacy. In this thesis, we show how to do so using a minimally trusted manager. Some works in the minimally trusted manager category include [6] [27] and [28].

In sMPC, multiple parties want to perform some joint computation without revealing their secret input values to one another. Instead of using a minimally trusted manager, every party performs a portion of the overall computation using their secret inputs in a manner that does not leak them [29] [30] [31] [32].

Lastly, there is the transaction-based approach that is similar to what is found in public blockchain frameworks, such as Ethereum. Each transaction triggers some on-chain public computation to be executed and can even update application-specific storage in some way. These works attempt to maintain the confidentiality of certain variables, storage, and information during each transaction's execution [33] [8] [34] [35]. We believe this approach will be the most difficult to add support for volume and timing privacy because there are many

blockchain properties that have to be accounted for, such as which storage slot gets updated or which portion of some code is executed.

2.2 Timing Metadata on Blockchains

Works like [36] have shown how to create multi-party commitments that can only be revealed after a certain time. While this can be used for multi-party applications like ours, the commitments are required to eventually be revealed in order to finalize the application, which eventually leaks volume and timing information. Our approach can privatize the volume and timing of transactions even after the application has been finalized.

[37] have shown how to create payment transactions that are reversible for a period of time. [38] have shown how to schedule transactions to be posted to the blockchain at some predefined time in the future. However, reversing transactions and delaying transactions are workarounds based on the fact that volume and timing information is public. They do not help to privatize the volume and timing of an application's transactions.

Works like [39] [40] have shown that you can create anonymous tamper-proof timestamps for off-chain digital content such as documents, papers, and videos. However, these approaches cannot be done anonymously and privately for on-chain transactions, only for off-chain hashes or other off-chain cryptographic commitments.

2.3 Privatizing Which Computation Was Performed

We do note that, in a framework where computations can be verifiably executed without leaking which computation was executed, there would be no need for a solution such as

ours because transactions could not be linked to a specific application. However, to our knowledge, there is no approach that can currently do this. [41] have attempted to hide which computation was performed, but their approach is not suited to maintain the integrity and verifiability of many blockchain applications.

2.4 Mitigating Through High Frequency

Volume and timing information may be rendered useless for some applications with a sufficiently large number of transactions. For example, on the ZCash blockchain, it is accepted that the timing and volume of each private and anonymous payment will be leaked, but since the volume of payments is sufficiently large, the volume and timing distribution of payments does not leak any useful information. At the time of writing this (June 17, 2022 7:53AM), the ZCash network has handled 10,678,018 payments and 154,306,966 payment outputs involving 799,215 unique addresses [42]. This equates to about 0.868 transaction outputs per second since its genesis block on October 28, 2016. However, most blockchain applications will not have the luxury of having such high transactional volume. Our approach is suited to privatize volume and timing information even in niche applications with unique functionalities and a very small number of transactions.

To demonstrate, consider at a hypothetical example where the ZCash network was sufficiently small such that the volume and timing metadata actually did leak information. Imagine the ZCash network only consisted of a building owner and the renters of that building, and the network is used to privately and anonymously pay rent each month to the building owner. Although payments could still be fully anonymous and privacy-preserving, one renter may be able to use the volume and timing metadata of all other payments to determine how many of the other renters' payments were missing or late that month.

Chapter 3

Proposed Method

3.1 Preliminaries

3.1.1 Notation

Let $||$ represent concatenation. We denote $[N] = \{1, ..., N\}$. Let a Probabilistic Polynomial Time (PPT) adversary be an adversary who is computationally bounded and can only execute PPT computations.

Symmetric Key Encryption

We denote $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ as symmetric key encryption where $k \leftarrow \mathcal{E}.\mathsf{Gen}(1^\lambda)$ generates a symmetric key k with a security parameter λ, $c \leftarrow \mathcal{E}.\mathsf{Enc}(k, m)$ encrypts plaintext m with the key k, and $m \leftarrow \mathcal{E}.\mathsf{Dec}(k, c)$ decrypts the ciphertext c with the key k.

Asymmetric Key Encryption

We denote $\mathcal{E}' = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ as asymmetric key encryption where $(\mathsf{pk}_{\mathsf{enc}}, \mathsf{sk}_{\mathsf{enc}}) \leftarrow \mathcal{E}'.\mathsf{Gen}(1^\lambda)$ generates a a unique key pair $(\mathsf{pk}_{\mathsf{enc}}, \mathsf{sk}_{\mathsf{enc}})$ with a security parameter λ, $c \leftarrow \mathcal{E}'.\mathsf{Enc}(\mathsf{pk}_{\mathsf{enc}}, m)$ encrypts plaintext m with the public key $\mathsf{pk}_{\mathsf{enc}}$, and $m \leftarrow \mathcal{E}'.\mathsf{Dec}(\mathsf{sk}_{\mathsf{enc}}, c)$ decrypts the ciphertext c with the secret key $\mathsf{sk}_{\mathsf{enc}}$.

Cryptographic Signatures

Let $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ represent cryptographic digital signatures where $(\mathsf{pk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{sig}}) \leftarrow \Sigma.\mathsf{Gen}(1^{\lambda})$ generates a key pair, $\sigma \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}_{\mathsf{sig}}, m)$ produces a signature σ for plaintext m using $\mathsf{sk}_{\mathsf{sig}}$, and $\{0, 1\} \leftarrow \Sigma.\mathsf{Verify}(\mathsf{pk}_{\mathsf{sig}}, m, \sigma)$ verifies whether the signature σ of m is valid (1) for $\mathsf{pk}_{\mathsf{sig}}$ or not (0).

Blockchain Transactions

We denote $\mathsf{Txn} = (\mathsf{Create}, \mathsf{Parse}, \mathsf{Timestamp})$ to represent select functionality of blockchain transactions. The evaluation $(\mathsf{txn}, \sigma) \leftarrow \mathsf{Txn.Create}(\mathsf{pk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{sig}}, m)$ constructs a well-formed blockchain transaction txn associated with $\mathsf{pk}_{\mathsf{sig}}$ where the transaction's contents consist of the message m and σ is the transaction's signature obtained from $\sigma \leftarrow \Sigma.\mathsf{Sign}(\mathsf{sk}_{\mathsf{sig}}, m)$. The evaluation $(\mathsf{pk}_{\mathsf{sig}}, m) \leftarrow \mathsf{Txn.Parse}(\mathsf{txn})$ parses a transaction txn to get its respective $\mathsf{pk}_{\mathsf{sig}}$ and m. Informally, $\mathsf{pk}_{\mathsf{sig}}$ can be thought of as a public pseudonym which represents who submitted a blockchain transaction. The evaluation $(\sigma, \mathsf{txn}, y) \leftarrow \mathsf{Txn.Timestamp}(\sigma, \mathsf{txn})$ assigns the current timestamp y to the transaction (σ, txn). The tuple $(\sigma, \mathsf{txn}, y)$ represents a complete blockchain transaction.

Zero-Knowledge Proofs and zkSNARKs

In this thesis, we use zero-knowledge Succinct Non-interactive ARguments of Knowledge (zkSNARKs) [43] [44]. zkSNARKs enable a prover $\mathcal{P}$ to generate a succinct proof π that can convince a verifier $\mathcal{V}$ that they know some NP statement is true without revealing anything other than the veracity of the statement [44] [45]. In other words, for an NP relation $\mathcal{R}$ and an NP language $\mathcal{L}$, $\mathcal{P}$ can convince $\mathcal{V}$ that it knows a witness $\mathbf{w}$ for some input $\mathbf{s} \in \mathcal{L}$. such that $\mathbf{s}, \mathbf{w} \in \mathcal{R}$ and $\mathcal{R}(\mathbf{s}, \mathbf{w}) = 1$. zkSNARKs are widely used within the blockchain industry

today [46] [47].

zkSNARKs satisfy the following properties:

- Completeness — $\mathcal{P}$ can convince $\mathcal{V}$ of any true statement
- Soundness — a malicious $\mathcal{P}$ can not convince $\mathcal{V}$ of a false statement
- Zero-knowledge — nothing else is revealed to $\mathcal{V}$ besides that the statement is true

Formally, the functionality of zkSNARKs consists of three algorithms as $\mathsf{zkp} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ [48]:

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R})$: Given a security parameter λ and an NP relation $\mathcal{R}$, the public parameters pp are generated.
- $\pi \leftarrow \mathsf{Prove}(\mathsf{pp}, \mathbf{s}, \mathbf{w})$: Given pp, a public statement $\mathbf{s}$, and a private witness $\mathbf{w}$ where $\mathbf{s}, \mathbf{w} \in \mathcal{R}$ and $\mathcal{R}(\mathbf{s}, \mathbf{w}) = 1$, a proof π is generated.
- $\{0, 1\} \leftarrow \mathsf{Verify}(\mathsf{pp}, \mathbf{s}, \pi)$: Given pp, the proof π, and the public statement $\mathbf{s}$ (but not $\mathbf{w}$), return if the proof is valid (1) or invalid (0).

3.2 Models

3.2.1 System Model

Our system consists of U participating users and a manager. We assume that the manager is not a participating user. Each user will have a secret unique user identifier uid only known to them. We will refer to specific users using uid. Note these identifiers are different from public blockchain pseudonyms because they are secret and never known by others.

In our system, each user and the manager communicate with each other through blockchain transactions in order to facilitate a multi-party application with a time-sensitive target function F as

$$\{(o_i, \mathsf{uid}'_i)\}_{i=1}^{Q} \leftarrow F(\{(\mathsf{uid}_j, x_j, y_j, z_j)\}_{j=1}^{N})$$

where N inputs are submitted to the target function F by the U users. o_i represents a secret output value for the application specific to user uid'_i. Q represents the number of outputs for F. Each input (uid_j, x_j, y_j, z_j) is a tuple where uid_j is the uid of the user submitting the input, x_j is the user's secret input value, y_j is the timestamp of the input assigned by the blockchain, and $z_j \in \{0, 1\}$ is the secret bit value denoting if the input is a decoy ($z_j = 0$) or real ($z_j = 1$), respectively (see Figure 3.1). We say the function F is time-sensitive because the output depends on the time (y_j) when users submit their inputs.

Our system consists of three stages Initialization, Submission, and Finalization spanning across the times T_0, T_1, T_2, and T_3 as follows.

- Initialization (T_0 to T_1): First, the application is initialized by defining the logic of function F, along with other setup to initialize system parameters.

- Submission (T_1 to T_2): During this stage, each user uid will provide one or more inputs to the common function F. To submit an input, they will send (uid_j, x_j, z_j) to the manager via a blockchain transaction. The blockchain then assigns a public timestamp y_j to this transaction. This creates the tuple (uid_j, x_j, y_j, z_j).

- Finalization (T_2 to T_3): In this stage, the manager privately executes the function logic F on all the submitted (uid_j, x_j, y_j, z_j) inputs and sends each o_i back to the corresponding user uid'_i via a blockchain transaction.

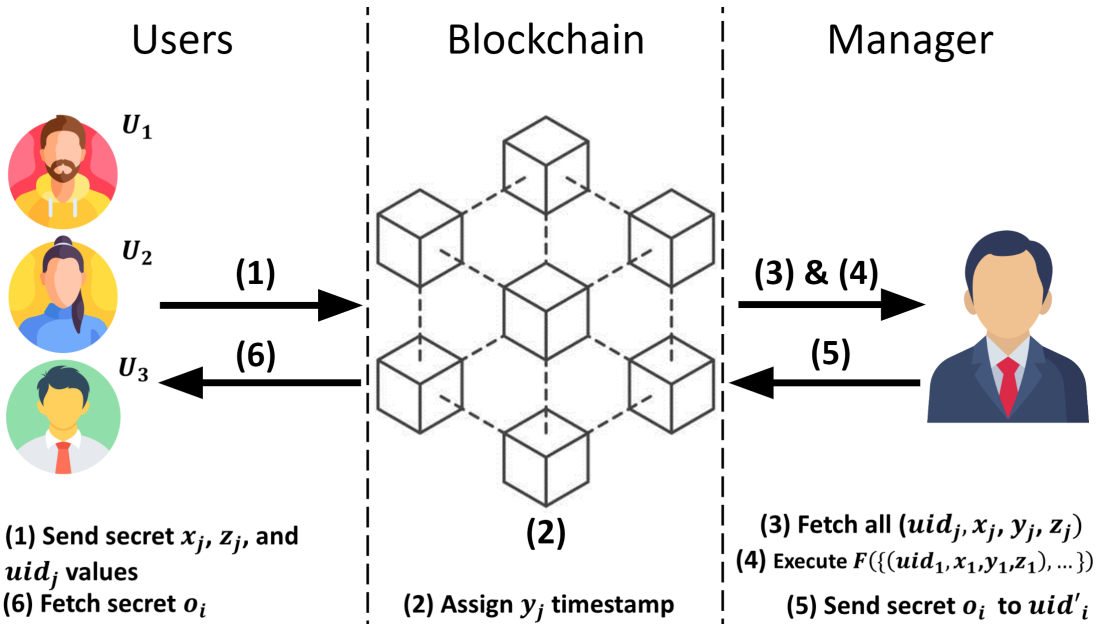

Figure 3.1: System model figure

3.2.2 Threat Model

In our system, the blockchain is trusted. Specifically, we trust that it will provide a correct timestamp y_j for the users' transactions, ensure data availability, and correctly verify if transactions are well-formed.

Similar to [6], we minimally trust the manager. Specifically, we trust the manager to maintain the privacy of all users' secret input values $(\mathsf{uid}_j,\ x_j,\ z_j)$ and secret output values (o_i, uid'_i). We also trust them to privately execute the logic of the target function F and send each secret o_i back to the corresponding user uid'_i. However, we do not trust they will do so correctly. We assume that they may attempt to deviate from the protocol (i.e. excluding/editing/adding inputs to F, executing a different computation F', or manipulating o_i values). We assume each user will maintain the privacy of their own secret $(\mathsf{uid}_j,\ x_j,\ z_j)$ and (o_i, uid'_i) values. However, we do not trust the user with correctly revealing $(\mathsf{uid}_j,\ x_j,\ z_j)$ to the manager.

We assume all blockchain users can be the adversaries that attempt to learn about the other

users' secret input values (uid_j, x_j, z_j) and secret output values (o_i, uid'_i) using the target function F and all blockchain transactions.

3.2.3 Security Guarantees

To enable security against the aforementioned adversaries, we aim to achieve three security properties: a) input value privacy, b) correctness of execution, and c) volume and timing privacy. The first two are inherited from [6], while volume and timing privacy is our new security feature.

Input Value Privacy: Input value privacy is achieved if a PPT adversary can not deduce the values of any secret input (uid_j, x_j, z_j) or output (o_i, uid'_i) unavailable to them.

Correctness of Execution: Even though the users and the manager may attempt to act maliciously and selfishly to maximize their own outcomes, the execution of the application is guaranteed to be correct, fair, and honest.

Volume and Timing Privacy: Volume and timing privacy is achieved if a PPT adversary can not learn the volume and timestamps of the real user inputs as well as their impact on the outcome of the target function F.

3.3 Proposed Method

3.3.1 Overview

In this section, we propose our method to privatize the volume and timing of transactions for blockchain applications in a manner that achieves all of our previously outlined security guarantees. We use a base approach from [6] that utilizes a minimally trusted manager,

verifiable zkSNARKs, encryptions, and uniquely generated pseudonyms for each user input submission transaction. We then extend this approach to privatize the volume and timestamps of an application's real inputs ($z_j = 1$) through the use of decoy inputs ($z_j = 0$). Decoy inputs never affect the application's outcome whereas real ones can. The only purpose of decoy inputs is to add another blockchain transaction to create additional obfuscation. To an adversary, decoy inputs will be kept indistinguishable from real inputs a) by keeping each z_j value encrypted and b) by the design of the target function F. An example distribution of real versus decoy transactions can be seen in Figure 3.2. To an adversary, these transactions all look the same as in Figure 3.3.

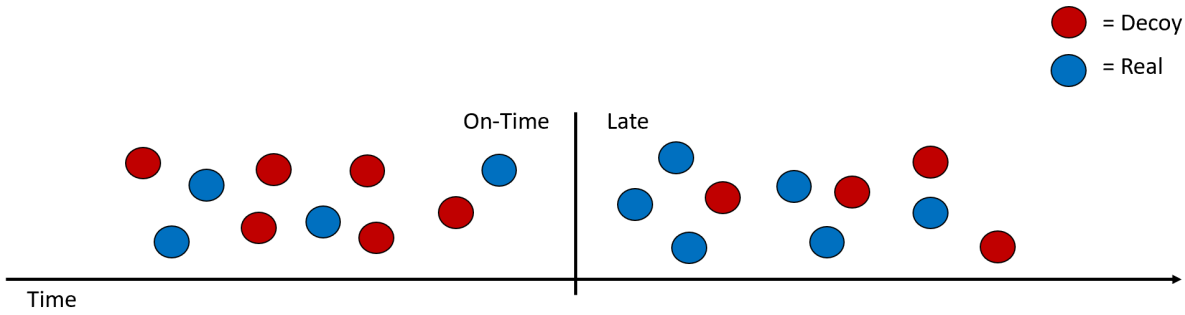

Figure 3.2: Example distribution of real versus decoy transactions for an application.

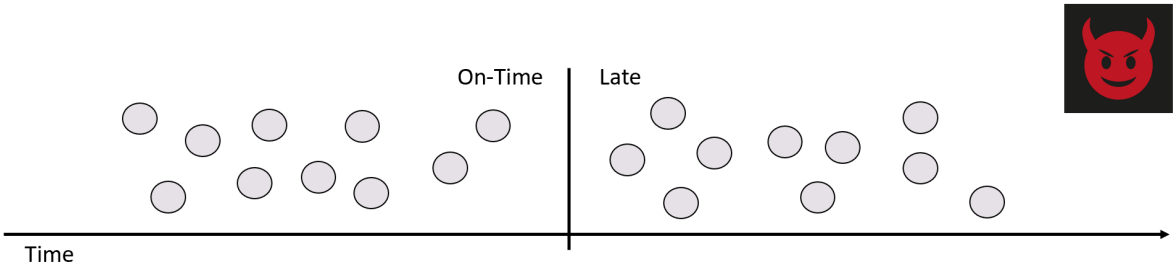

Figure 3.3: An adversary's view of the distribution from Figure 3.2.

The intuition behind our approach is that if sufficient noise is added through indistinguishable decoy inputs, a PPT adversary will not be able to determine if any user input submission transaction corresponds to a real input or a decoy input, given that they have access to F,

the N input submission transactions, and the manager's finalization transaction. Thus, even though the volume and timestamps of all transactions can be publicly observed on the blockchain, the exact volume and timestamps of the application's real inputs cannot be learned, even with state-of-the-art solutions.

The challenge we face is that even if each z_j can be kept private using the manager, the application-specific design of F in combination with the volume and timing distribution of the N submission transactions oftentimes will leak information to an adversary. For example, suppose we have a deadline-based application (e.g. a student exam) where inputs are deemed on-time or late according to a public deadline during Submission (i.e. deadline is some time between T_1 and T_2). If there is only one input submitted before the deadline, an adversary can observe the blockchain to learn this (even if z_j is kept private), and this will leak that $\geq U - 1$ users submitted late. Another example is that an adversary could guess that submissions posted close to the deadline are more likely to be real than decoy due to procrastination. Adversaries will constantly adapt to recognizing new volume and timing patterns to learn information.

3.3.2 The Application's Target Function

Designing F

While the target function F and determining the outcome of an application is application-specific, we enforce F to follow a specific structure that protects against an adversary analyzing the volume and timing distribution of the N submission transactions.

Let there be K timestamps $\{t_1, t_2, \ldots t_K\}$ where $T_1 \leq t_1 \leq \ldots \leq t_K \leq T_2$ and let Δ be a short length of time. Let g represent the portion of logic within F that is application-specific and

determines the application's outcome using only the real and valid inputs as

$$\{(o_i, \mathsf{uid}'_i)\}_{i=1}^{Q} \leftarrow g(\{(\mathsf{uid}_d, x_d, y_d, z_d)\}_{d=1}^{H})$$

where $H \leq N$. Within F, we first filter the original inputs $\{(\mathsf{uid}_j, x_j, y_j, z_j)\}_{j=1}^{N}$ down to H inputs by removing decoy inputs and removing inputs from users who do not submit at all K time periods ($t \pm \Delta$). Then, the real and valid inputs (i.e. the filtered H inputs) are inputted to g to compute the outcome $\{(o_i, \mathsf{uid}'_i)\}_{i=1}^{Q}$. Formally, the logic of $\{(o_i, \mathsf{uid}'_i)\}_{i=1}^{Q} \leftarrow F(\{(\mathsf{uid}_j, x_j, y_j, z_j)\}_{j=1}^{N})$ is structured as:

- $\mathcal{I} \leftarrow \{(\mathsf{uid}_j, x_j, y_j, z_j)\}_{j=1}^{N}$
- $\mathcal{U} \leftarrow \{\emptyset\}$
- For each $j \in [N]$, if $\mathsf{uid}_j \notin \mathcal{U}$ and for each $a \in [K]$, $\exists\ (\mathsf{uid}_b, x_b, y_b, z_b) \in \mathcal{I}$ such that $b \in [N]$, $\mathsf{uid}_b = \mathsf{uid}_j$, and $t_a - \Delta \leq y_b \leq t_a + \Delta$: $\mathcal{U} \leftarrow \mathcal{U} \cup \{(\mathsf{uid}_j)\}$. This is to filter out the inputs from users who do not submit during all K time periods.
- $\mathcal{I}' \leftarrow \{\emptyset\}$
- For each $j \in [N]$, if $z_j = 1$ and $\mathsf{uid}_j \in \mathcal{U}$: $\mathcal{I}' \leftarrow \mathcal{I}' \cup \{(\mathsf{uid}_j, x_j, y_j, z_j)\}$. This is to filter out the decoy inputs.
- $\{(o_i, \mathsf{uid}'_i)\}_{i=1}^{Q} \leftarrow g(\mathcal{I}')$

The inputs that are not submitted during any $t_i \pm \Delta$ are never passed to F because their corresponding transactions will be publicly rejected by the blockchain (see Section 3.3.3). Through the above logic, only real inputs from users who submit during all K time periods can affect the application's outcome $\{(o_i, \mathsf{uid}'_i)\}_{i=1}^{Q}$.

Length of Δ

Ideally, Δ would equal 0 and all submissions are synchronized, but this is not possible because transactions on a blockchain can never be guaranteed to be added at a specific time. In our system, Δ should be as short as possible but also long enough that users will not fall victim to denial-of-service attacks or be unlucky where their transaction is not confirmed in time by the asynchronous blockchain.

Time-Based and Volume-Based Outcomes

For applications with time-sensitive or volume-sensitive outcomes (e.g. a student exam or a Dutch auction), the K time periods should span all possible volume and timing combinations that could potentially affect any o_i. For example, in a deadline-based application (as discussed in Section 3.3.1), the o_i values are dependent on if the y_j input timestamps are either before or after some public deadline, so, in this case, a minimum of K=2 time periods are required: one before and one after the deadline (see Figure 3.4). For a Dutch auction, if the price is decremented Y times, there should be K=Y time periods where the time periods span all possible prices. This will result in a distribution where volume and timing privacy is achieved, and adversaries can not deduce anything from the volume and timing distribution of the application's transactions (see Figure 3.5).

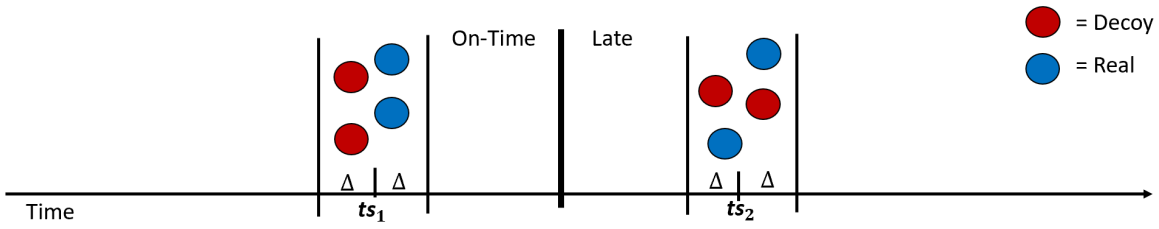

Figure 3.4: A deadline-based application using our approach with K=2 time periods.

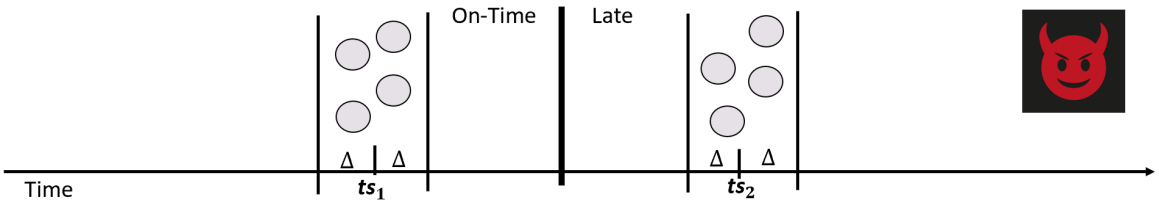

Figure 3.5: An adversary's view of the distribution in Figure 3.4.

3.3.3 Detailed Algorithm

In this section, we present our algorithm in detail. Let $\mathcal{R}$ be the relation for zkp where users generate a proof π that they encrypted their secret $(\mathsf{uid}_j, x_j, z_j)$ values to the manager correctly. Let $\mathcal{R}'$ be the relation for zkp where the manager generates a proof π' that they finalized the application correctly. This consists of a) proving knowledge of all users' secret $(\mathsf{uid}_j, x_j, z_j)$ values, b) proving F was executed correctly, and c) proving the secret o_i output values were encrypted and sent correctly to each user uid'_i via the blockchain. We utilize both symmetric $\mathcal{E}$ and asymmetric $\mathcal{E}'$ encryptions during our algorithm so that generating π' can be done more efficiently using only $\mathcal{E}$ rather than $\mathcal{E}'$ (see Section 4).

1) $\mathsf{Initialization}$ (T_0 to T_1): First, the manager needs to establish their key pairs, initialize the application's functionality, and initialize all system parameters as:

- $(\mathsf{pk}_{\mathsf{sig}_\mathsf{M}}, \mathsf{sk}_{\mathsf{sig}_\mathsf{M}}) \leftarrow \Sigma.\mathsf{Gen}(1^\lambda)$
- $(\mathsf{pk}_{\mathsf{enc}_\mathsf{M}}, \mathsf{sk}_{\mathsf{enc}_\mathsf{M}}) \leftarrow \mathcal{E}'.\mathsf{Gen}(1^\lambda)$
- $\mathsf{pp} \leftarrow \mathsf{zkp.Setup}(1^\lambda, \mathcal{R})$
- $\mathsf{pp}' \leftarrow \mathsf{zkp.Setup}(1^\lambda, \mathcal{R}')$
- Broadcast $\mathsf{pk}_{\mathsf{sig}_\mathsf{M}}$, $\mathsf{pk}_{\mathsf{enc}_\mathsf{M}}$, F, and any other application-specific logic on the blockchain (including the parameters of pp and pp' which the blockchain will need during $\mathsf{Submission}$

and Finalization to execute zkp.Verify).

- Broadcast pp to all users. The users will locally store pp. We recommend IPFS (InterPlanetary File System) for availability and tramper-proof storage [49].

Each user establishes a unique symmetric encryption key:

- $k \leftarrow \mathcal{E}.\mathsf{Gen}(1^\lambda)$

2) Submission (T_1 to T_2): For a user to submit an input to the target function F, they perform the following operations:

- Generate secret values for x and z.
- $c \leftarrow \mathcal{E}.\mathsf{Enc}(k, z||x)$
- $c' \leftarrow \mathcal{E}'.\mathsf{Enc}(\mathsf{pk}_{\mathsf{enc_M}}, k)$
- $\mathbf{w} \leftarrow (x,\ z,\ k)$
- $\mathbf{s} \leftarrow (c,\ c',\ \mathsf{pk}_{\mathsf{enc_M}})$
- $\pi \leftarrow \mathsf{zkp.Prove}(\mathsf{pp},\ \mathbf{s},\ \mathbf{w})$
- $(\mathsf{pk}_{\mathsf{sig}}, \mathsf{sk}_{\mathsf{sig}}) \leftarrow \Sigma.\mathsf{Gen}(1^\lambda)$
- $(\mathsf{txn},\ \sigma) \leftarrow \mathsf{Txn.Create}(\mathsf{pk}_{\mathsf{sig}},\ \mathsf{sk}_{\mathsf{sig}},\ \pi\ ||\ \mathbf{s})$
- Send $(\mathsf{txn},\ \sigma)$ to the blockchain

Upon the blockchain receiving each tuple (txn, σ), the blockchain will execute:

- $(\mathsf{txn}, \sigma, y) \leftarrow \mathsf{Txn.Timestamp}(\mathsf{txn}, \sigma)$

- $(\mathsf{pk}_{\mathsf{sig}}, \pi \mid\mid \mathbf{s}) \leftarrow \mathsf{Txn.Parse}(\mathsf{txn})$
- $(c, c', \mathsf{pk}_{\mathsf{enc_M}}) \leftarrow \mathbf{s}$
- If $\{0,1\} \leftarrow \Sigma.\mathsf{Verify}(\mathsf{pk}_{\mathsf{sig}}, \mathsf{txn}, \sigma) = 0$ or $\{0,1\} \leftarrow \mathsf{zkp.Verify}(\mathsf{pp}, \mathbf{s}, \pi) = 0$ or $\mathsf{pk}_{\mathsf{enc_M}}$ does not match what was defined in Initialization or y does not fall within one of the K time periods outlined in F, reject the transaction.
- Else, include $(\mathsf{txn}, \sigma, y)$ on the blockchain.

3) Finalization (T_2 to T_3): Assume there are N transactions $(\mathsf{txn}_j, \sigma_j, y_j)$ submitted by users during Submission, the manager executes the following for each $j \in [N]$:

- $(\mathsf{pk}_{\mathsf{sig}_j}, \pi_j \mid\mid \mathbf{s}_j) \leftarrow \mathsf{Txn.Parse}(\mathsf{txn}_j)$
- $(c_j, c'_j, \mathsf{pk}_{\mathsf{enc_M}}) \leftarrow \mathbf{s}_j$
- $k_j \leftarrow \mathcal{E}'.\mathsf{Dec}(\mathsf{sk}_{\mathsf{enc_M}}, c'_j)$
- $z_j || x_j \leftarrow \mathcal{E}.\mathsf{Dec}(k_j, c_j)$

The manager now uses all the decrypted input values to finalize the application. We will use each user's encryption key k from Initialization as their user identifier uid.

- $\{(o_i, k'_i)\}_{i=1}^{Q} \leftarrow F(\{(k_j, x_j, y_j, z_j)\}_{j=1}^{N})$
- $o'_i \leftarrow \mathcal{E}.\mathsf{Enc}(k'_i, k'_i \mid\mid o_i)$ for each $i \in [Q]$
- $\mathbf{w}' \leftarrow (\{(k_j, x_j, z_j)\}_{j=1}^{N}, \{(o_i, k'_i)\}_{i=1}^{Q})$
- $\mathbf{s}' \leftarrow (\{(c_j, y_j)\}_{j=1}^{N}, \{(o'_i)\}_{i=1}^{Q})$
- $\pi' \leftarrow \mathsf{zkp.Prove}(\mathsf{pp}', \mathbf{s}', \mathbf{w}')$

- $(\mathsf{txn}', \sigma') \leftarrow \mathsf{Txn.Create}(\mathsf{pk}_{\mathsf{sig}_\mathsf{M}}, \mathsf{sk}_{\mathsf{sig}_\mathsf{M}}, \pi' \,||\, \{(o_i')\}_{i=1}^{Q})$
- Send (txn', σ') to the blockchain

Upon receiving (txn', σ'), the blockchain verifies the application's finalization by executing:

- $(\mathsf{txn}', \sigma', y') \leftarrow \mathsf{Txn.Timestamp}(\mathsf{txn}', \sigma')$
- $(\mathsf{pk}_{\mathsf{sig}_\mathsf{M}}, \pi' \,||\, \{(o_i')\}_{i=1}^{Q}) \leftarrow \mathsf{Txn.Parse}(\mathsf{txn}')$
- Fetch $(\mathsf{txn}_j, \sigma_j, y_j)$ from past blocks for $j \in [N]$
- $(\mathsf{pk}_{\mathsf{sig}_j}, \pi_j \,||\, \mathbf{s}_j) \leftarrow \mathsf{Txn.Parse}(\mathsf{txn}_j)$ for $j \in [N]$
- $(c_j, c_j', \mathsf{pk}_{\mathsf{enc}_\mathsf{M}}) \leftarrow \mathbf{s}_j$ for $j \in [N]$
- $\mathbf{s}' \leftarrow (\{(c_j, y_j)\}_{j=1}^{N}, \{(o_i')\}_{i=1}^{Q})$
- If $\{0,1\} \leftarrow \Sigma.\mathsf{Verify}(\mathsf{pk}_{\mathsf{sig}_\mathsf{M}}, \mathsf{txn}', \sigma') = 0$ or $\{0,1\} \leftarrow \mathsf{zkp.Verify}(\mathsf{pp}', \mathbf{s}', \pi') = 0$ or $\mathsf{pk}_{\mathsf{sig}_\mathsf{M}}$ does not match the one defined in Initialization, reject the transaction.
- Else, include $(\mathsf{txn}', \sigma', y')$ on the blockchain.

Each user now has access to their respective output values by:

- Fetch $(\mathsf{txn}', \sigma', y')$ from the blockchain
- $(\mathsf{pk}_{\mathsf{sig}_\mathsf{M}}, \pi' \,||\, \{(o_i')\}_{i=1}^{Q}) \leftarrow \mathsf{Txn.Parse}(\mathsf{txn}')$
- $k_i' \,||\, o_i \leftarrow \mathcal{E}.\mathsf{Dec}(k, o_i')$ for $i \in [\mathrm{Q}]$
  - If $k_i' = k$, this output is for them.

3.3.4 Security Analysis

Input Value Privacy

To achieve input value privacy, we always keep each user's secret (uid_j, x_j, z_j) and (o_i, uid'_i) values encrypted on-chain between them and the manager, who is trusted for privacy, even after the application has been finalized. The manager never leaks such values during Finalization because the execution of F and encrypting each o_i to uid'_i (i.e. k'_i) is done privately but verifiably via π'. Additionally, users generate unique blockchain pseudonyms ($\mathsf{pk}_{\mathsf{sig}}$) for every blockchain transaction so that no two transactions can be publicly linked.

The above techniques to achieve input value privacy were inherited from [6], but our newly proposed approach of enforcing all U users to submit at K time periods improves upon these techniques. This is because as long as Δ is sufficiently short, our approach adds sufficient noise such that an adversary can not deduce any secret value using the volume and timing distribution of the N submission transactions (see Figure 3.4). For example, previously without any time periods, an adversary may have been able to guess that inputs submitted by users close to a deadline are more likely to be real ($z_j = 1$) than decoy ($z_j = 0$) due to procrastination.

Correctness of Execution

The application's correctness of execution is verified by the trusted blockchain to be correct, fair, and honest through verifying a) π from each user, which proves all inputs were revealed to the manager correctly, and b) π' from the manager, which proves the finalization of the application was correct. Due to lack of space, our algorithm did not explicitly handle the aborting manager case (i.e. no valid π' provided by T_3), but our approach can be extended

by enforcing a financial penalty if the manager aborts, as demonstrated in [6].

Additionally, we use the public y_j timestamps assigned by the blockchain (the trusted timekeeper). This is important to maintain the integrity of our time-sensitive target function F.

Volume and Timing Privacy

We maintain volume and timing privacy through the use of decoy inputs. Decoy inputs are kept indistinguishable from real inputs to an adversary as explained in Section 3.3.4. Decoy inputs never affect the application's outcome because. within F, they are filtered out and never get passed to g, which is used to calculate the secret application-specific output $\{(o_i, \mathsf{uid}'_i)\}_{i=1}^{Q}$. F is also executed privately but verifiably via π'. Because of these properties, an adversary can not learn the volume and timestamps of the real transactions which impacted the outcome of F (see Figure 3.4).

For applications with time-sensitive or volume-sensitive outcomes, we set the K time periods to span all possible volume and timing combinations that could affect any o_i (see Section 3.3.2). During all such time periods, we enforce all U users to submit at least one input, and these inputs can not be distinguished to be decoy or real by an adversary. This adds sufficient noise such that an adversary can not deduce any secret time-sensitive or volume-sensitive output o_i using F and the volume and timing distribution of the N submission transactions (see Figure 3.4 and 3.5).

User Amount Privacy

Lastly, we also want to note that if U is not explicitly public, an adversary can not learn the exact value of U due to the indistinguishability of decoy inputs, unique pseudonyms

being generated by users, and no maximum on the number of inputs submitted per user per each of the K time periods. For example, in Figure 3.4, an adversary can observe there is a maximum of eight users, but in practice, there can be anywhere from one to eight users.

Chapter 4

Experiment

4.1 Implementation

We implemented our proposed method in Solidity (i.e. the programming language of the Ethereum blockchain), JavaScript, and Java in about 4000 lines of code. First, we generated our zkSNARK circuit files and circuit input files with xjsnark [50] for the user's submission proof and manager's finalization proof. We term these circuits $\mathsf{zkSubmit}$ and $\mathsf{zkFinalizeN}$ where N is the total number of user inputs submitted during an application.

For symmetric key encryption, we used AES. For asymmetric key encryption, we used RSA. We used the RSA and AES gadgets provided by xjsnark [50]. The setup ($\mathsf{zkp.Setup}$) and proof generation ($\mathsf{zkp.Prove}$) was done with the libsnark and jsnark libraries [51] [52]. Specifically, we built libsnark with the MULTICORE (multi-threaded option) and ALT_BN_128 flags turned on and ran it with the Groth16 proof system option [53].

Within $\mathsf{zkFinalizeN}$, the manager proves knowledge of each user's inputs using AES encryption as $c \leftarrow \mathcal{E}.\mathsf{Enc}(k, z||x)$. This is more efficient within zkSNARKs compared to RSA decryption $k \leftarrow \mathcal{E}'.\mathsf{Dec}(\mathsf{sk}_{\mathsf{enc_M}}, c')$ and then AES decryption $z||x \leftarrow \mathcal{E}.\mathsf{Dec}(k, c)$ which is what they execute during $\mathsf{Finalization}$ to originally get the users' inputs.

For our blockchain, we used a local Ethereum testnet using the HardHat framework and Geth consisting of 10 Ethereum nodes initialized with a blank genesis state and producing

blocks every ∼12 seconds [54] [55].

We exported each zkSNARK circuit's verification logic to Solidity using EthSnarks which uses a modified version of Groth16 designed for Ethereum [56] [53]. Each circuit's verification logic (zkp.Verify) was deployed as its own Solidity library on our blockchain with one function: verifyProof.

We then deployed two Solidity smart contracts that implemented our proposed method: a generic contract and a Dutch auction contract. For both applications, N was equal to Q since we returned an output corresponding to every input. The generic contract had no application-specific logic and returned an output o_i for each user input where o_i was simply the respective input's y_j timestamp. The Dutch auction contract simulated a Dutch auction environment where a collection of J items were for sale. These items started at some publicly known price, P. At the end of each of the K time periods, the price of the items decreased publicly by P / K. All bidders submitted one real or decoy bid every time period (K). Upon finalization, we returned an output o_i for each bid where o_i was whether the bid was successful or not. A bid was successful if it was not a decoy bid and the items for sale had not sold out yet.

Both contracts had three functions to represent the three stages in our proposed method: Initialize, SubmitInput, and Finalize. Initialize initialized all deadlines, declared who the manager was, and defined the application's logic. SubmitInput verified the user's submission proof with verifyProof from the zkSubmit library and then stored the AES + RSA ciphertexts and the timestamp to storage. Finalize verified the manager's finalization proof with verifyProof from the respective zkFinalizeN library and asserted the AES ciphertexts and timestamps used in the proof exactly matched the ones previously stored. The architecture of our implementation can be seen in Figure 4.1.

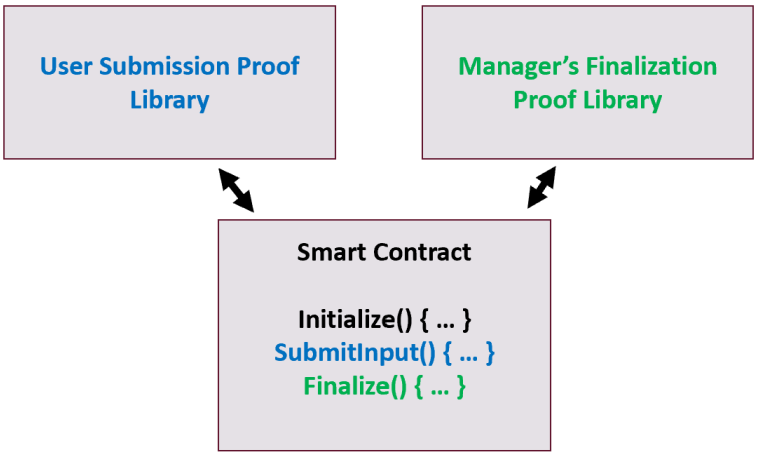

Figure 4.1: Architecture diagram for our implementation.

4.2 Hardware

For the user and blockchain, we used a desktop with a 12-core 12th Gen Intel Core i5-12400, 32 GB RAM, and a 1 TB SSD. For the manager, we used a server with a 48-core Intel Xeon Platinum 8360Y CPU @ 2.40 GHz, 128 GB RAM, and a 1.6TB SSD.

4.3 Parameters

We tested each application with the total number of user inputs $N = 8$, 16, 32, 64, and 128. For encryptions, we used a 128-bit AES key (128-bit security) and a 2048-bit RSA modulus (112-bit security). We used the ALT_BN128 elliptic curve for our zkSNARKs (100-bit security) [57] [58].

4.4 Metrics

We measured the Ethereum gas used, Ethereum storage used, and end-to-end delay (local proof generation + blockchain confirmation time). For the zkSNARK circuits, we reported the proving/verification key size, proof size, constraints, public inputs, and proving/verification/keygen time. The proving and verification keys are obtained from pp and are used for zkp.Prove and zkp.Verify, respectively. Constraints correspond to the size of the circuit. Public inputs correspond to the size of the statement $\mathbf{s}$.

4.5 Results

4.5.1 Generic Application

Table 4.1: Local overhead for our zkSNARK circuits (i.e. proof and key generation).

	Circuit Size		zkp.Setup			zkp.Prove	
Circuit	Constraints	Public Inputs	Time (s)	PKey (MB)	VKey (KB)	Proof (KB)	Time (s)
zkSubmit	105775	69	2.30	20.58	2.91	2.34	0.84
zkFinalize8	226568	73	2.99	33.65	3.07	2.46	1.14
zkFinalize16	453136	145	5.67	67.30	5.94	4.77	2.20
zkFinalize32	906272	289	8.08	134.60	11.68	9.375	4.33
zkFinalize64	1812544	577	14.35	269.21	23.17	18.59	8.62
zkFinalize128	3625088	1153	34.48	538.42	46.14	37.02	17.07

To evaluate, we generated and deployed our verification libraries for each of our circuits (Table 4.1 and 4.2). Then, we simulated the roles of all users and the manager for all three stages from our algorithm (Table 4.3).

Circuit Setup and Library Deployment: Table 4.1 and 4.2 shows the costs of setting up our circuits and deploying our verification libraries on the blockchain. Once deployed, our libraries can be reused across different applications' contracts without having to be redeployed. The zkSubmit circuit's constraints were dominated by the one RSA (∼89000 constraints) and

Table 4.2: Blockchain overhead for our zkSNARK circuits (i.e. verification and library generation).

	Circuit Size		zkp.Verify	Verification Library	
Circuit	Constraints	Public Inputs	Time (ms)	Deploy Gas	Bytecode (KB)
zkSubmit	105775	69	4.8	2851586	12.648
zkFinalize8	226568	73	4.6	2963548	13.15
zkFinalize16	453136	145	4.6	4994336	22.33
zkFinalize32	906272	289	4.6	9103364	40.88
zkFinalize64	1812544	577	4.7	18119193	81.67
zkFinalize128	3625088	1153	9.7	35333875	159.268

Table 4.3: Blockchain overhead for our proposed method with N Inputs (N = Real + Decoy).

	Initialization		Submission		Finalization	
N Inputs	Gas	Bytecode (KB)	Gas	Storage (KB)	Gas	Storage (KB)
8	1270118	4.28	19005140	36.384	988195	2.46
16	1270118	4.28	37993180	72.768	1737700	4.77
32	1270130	4.28	75969260	145.536	3237560	9.375
64	1270130	4.28	151921420	291.072	6240330	18.59
128	1270130	4.28	303825740	582.144	12258474	37.02

one AES encryption (∼14150 constraints). The zkFinalizeN circuits' constraints were dominated by the $2N$ AES encryptions (i.e. for proving knowledge of the N inputs and for encrypting each of the $N = Q$ outputs). This is why the metrics for these circuits grow linearly as N grows. While our circuits had minor helper functions like bit and array conversions, these only contributed a small amount of constraints in comparison to the expensive RSA and AES encryptions.

For our zkFinalize8 circuit, key generation takes 2.99 seconds, and the generated proving and verification key sizes are 33.65 MB and 3.07 KB, respectively. While the proving key is large, this is stored locally. On the other hand, the verification key is needed to be entirely stored on-chain within our libraries to implement verifyProof. These libraries only needed to store the verification key from pp.

Due to the verification keys being kilobytes in size, our libraries' bytecode sizes are largely

dominated by the storage of the verification key. For example, the storage of the verification key alone takes up 96% of our lines of Solidity code for the zkFinalize128 library.

The deployment gas calculation of the library is dominated by the bytecode size because Ethereum charges 200-264 gas per each byte of deployed code (depending on the number of zero bytes). There are other minor gas costs such as a fixed cost of 53000 gas per deployment, but for our large libraries, these other costs are very small in comparison.

Deployment of our libraries gets more expensive when N increases. For the zkFinalize32, zkFinalize64, and zkFinalize128 libraries, deployment exceeded predefined Ethereum mainnet network limits (i.e. 30,000,000 gas limit per block and 24.576 KB bytecode size) [59]. Ethereum does recognize these limits are low for applications like ours using zkSNARKs and are working on solutions such as EIP-196 to allow zkSNARKs on Ethereum to be implemented more efficiently [60].

Overhead per Stage: Table 4.3 shows the blockchain overhead for each of our three stages from our proposed method. For $N = 8$, the initialization stage (deploying the contract and calling Initialize) cost 1270118 gas and used 4.28 KB storage. This cost was almost the exact same for all N values because the only small change needed in the code was editing which zkFinalizeN library to call.

Before calling SubmitInput or Finalize, local proof generation is required. For the user's submission proof, this took 0.84 seconds. For the manager's finalization proof, this took from 1-17 seconds depending on N. Verification of these proofs only took 4-9 milliseconds.

Each user's execution of SubmitInput cost 2373505 gas and used 4.548 KB of storage. This storage number includes the 2.34 KB proof and 2.208 KB for storing the RSA ciphertext, AES ciphertext, and the timestamp to the contract's storage (which consisted of 69 uint256 integers). The overhead for the submission stage increases linearly with N because applica-

tions with more inputs require more submissions.

The manager's execution of Finalize when $N = 8$ cost 988195 gas and used 2.46 KB storage (finalization proof was 2.46 KB). This grows linearly with N due to the increasing proof size.

We do note some applications may not require all inputs to be used together in the same computation. For example, students' exam grades may have no dependence on one another. In cases like that, the manager could generate up to N separate π' proofs. This may be a more optimal solution, especially if the manager's finalization transaction exceeds predefined blockchain limits.

4.5.2 Dutch Auction

In this section, we show how our system performed when used by a real-world application: a Dutch auction. We tested Dutch auctions with $K = 1$, 2, and 4 time periods and up to 128 bids. We set the number of items for sale to $J = 8$.

Figure 4.2 shows that the total end-to-end delay for each bidder only depends on the number of bids they submit and is independent of how many other bidders there are. We extended Figure 4.2 and Figure 4.3 using projected values to better display our results. When each bidder only submits one bid ($K = 1$), it takes around seven seconds on average ($\sim$0.8 seconds for proof generation + $\sim$6 seconds for blockchain confirmation). When each bidder submitted two and four bids, the delay increases linearly to $\sim$14 and $\sim$27 seconds.

Blockchain confirmation time converged to an average of six seconds as we ran the experiment more times (25 runs in total). This is because broadcasting the pending transaction to all nodes only took a few milliseconds, but blocks were confirmed every 12 seconds. So in practice, each confirmation was effectively a random value between 0 and 12 seconds.

Figure 4.3 shows that the total delay for the auctioneer is directly correlated to the number of bids. The end-to-end delay is initially dominated by the blockchain confirmation time for small N values. However, the manager's proof generation time and thus end-to-end delay grows linearly with more bids. This is why the end-to-end delay doubles from $K = 1$ to $K = 2$ as well as from $K = 2$ to $K = 4$.

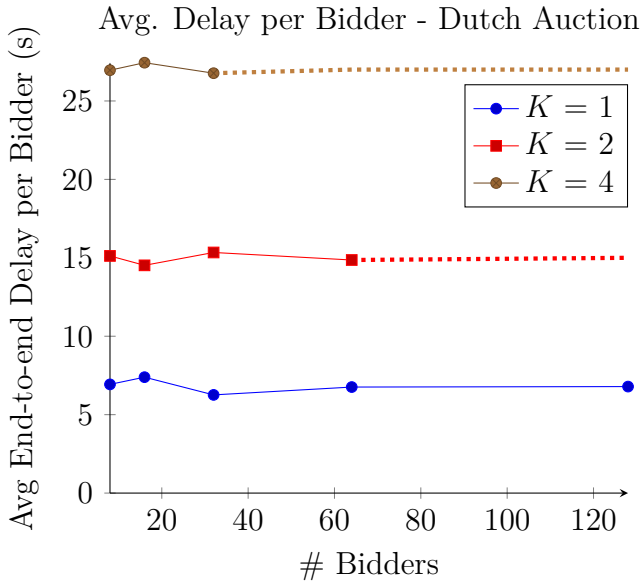

Figure 4.2: Average end-to-end time delay (proof generation + blockchain confirmation) per bidder in a Dutch auction.

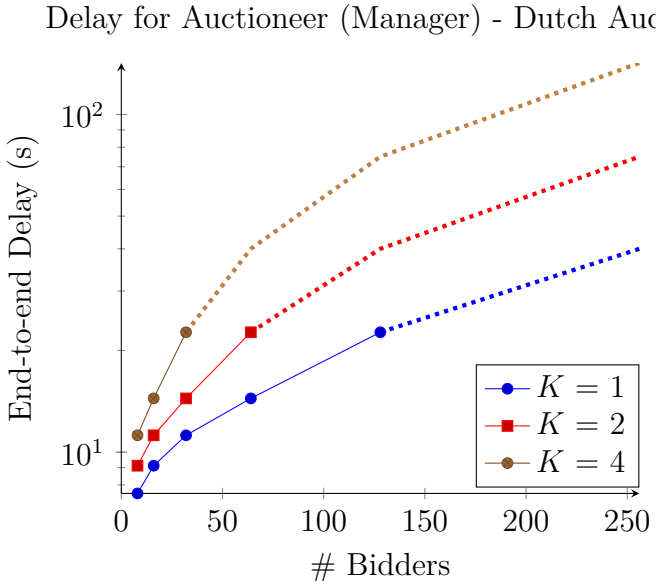

Figure 4.3: End-to-end time delay (proof generation + blockchain confirmation) for an auctioneer in a Dutch auction.

Chapter 5

Conclusion

In this thesis, we propose a new approach to implementing multi-party privacy-preserving blockchain applications such that the volume and timestamps of these applications' transactions never leaked any private information, even if the application is time-sensitive or volume-sensitive. This has now enabled these time-sensitive or volume-sensitive applications to be implemented on blockchains in a privacy-preserving manner, which was previously not possible with state-of-the-art solutions.

We achieved this without sacrificing the application's functionality, integrity, or verifiability through verifiable zkSNARKs and encryptions. Due to the core architecture of blockchains, privatizing the timing and volume information for an application could not be done directly because blockchain contents are stored and updated in a distributed manner. Our solution worked around this by having applications be implemented in a way that supports decoy, no-op transactions that are indistinguishable from real transactions. We showed how to design an application such that sufficient obfuscation is added through these decoy transactions such that adversaries could not deduce which transactions actually impacted the outcome of the application. As a result, adversaries could not learn anything about a volume-sensitive or time-sensitive application's outcome, even if the volume and timing distribution of transactions was publicly available. Future work can research optimizations to our approach and supporting decoy transactions for other categories for other categories of blockchain applications.

Bibliography

[1] Tharaka Mawanane Hewa, Yining Hu, Madhusanka Liyanage, Salil S. Kanhare, and Mika Ylianttila. Survey on blockchain-based smart contracts: Technical aspects and future research. *IEEE Access*, 9:87643–87662, Mar 2021.

[2] Ashutosh Ranade and Zaheed Shaikh. A survey on blockchain technology with use-cases in governance. *SSRN Electronic Journal*, Apr 2020.

[3] Fran Casino, Thomas K. Dasaklis, and Constantinos Patsakis. A systematic literature review of blockchain-based applications: Current status, classification and open issues. *Telematics and Informatics*, 36:55–81, 2019.

[4] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[5] Li Peng, Wei Feng, Zheng Yan, Yafeng Li, Xiaokang Zhou, and Shohei Shimizu. Privacy preservation in permissionless blockchain: A survey. *Digital Communications and Networks*, 7(3):295–307, 2021.

[6] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. *2016 IEEE Symposium on Security and Privacy (SP)*, 2016.

[7] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. *2014 IEEE Symposium on Security and Privacy*, 2014.

[8] Samuel Steffen, Benjamin Bichsel, Mario Gersbach, Noa Melchior, Petar Tsankov, and

Martin Vechev. Zkay. *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019.

[9] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. Zexe: Enabling decentralized private computation. *2020 IEEE Symposium on Security and Privacy (SP)*, 2020.

[10] John Rooksby and Kristiyan Dimitrov. Trustless education? a blockchain system for university grades1. *Ubiquity: The Journal of Pervasive Media*, 6(1):83–88, 2019.

[11] Zeshun Shi, Cees de Laat, Paola Grosso, and Zhiming Zhao. When blockchain meets auction models: A survey, some applications, and challenges, 2021.

[12] U.S. Department of Education. Family educational rights and privacy act (ferpa), Aug 2021.

[13] Ethereum Organization. Maximal extractable value (mev).

[14] Ivan Bogatyy. 0x605 100 cryptopunks, Aug 2021.

[15] Larva Labs. Cryptopunks.

[16] Flashbots. Mev explore.

[17] Ekin Genç. What is mev, aka maximal extractable value?, Sep 2022.

[18] Hisham S. Galal and Amr M. Youssef. Verifiable sealed-bid auction on the ethereum blockchain. *Financial Cryptography and Data Security*, page 265–278, 2019.

[19] Gaurav Sharma, Denis Verstraeten, Vishal Saraswat, Jean-Michel Dricot, and Olivier Markowitch. Anonymous fair auction on blockchain. In *2021 11th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1–5, 2021.

[20] Po-Chu Hsu and Atsuko Miyaji. Verifiable m+lst-price auction without manager. In *2021 IEEE Conference on Dependable and Secure Computing (DSC)*, pages 1–8, 2021.

[21] Po-Chu Hsu and Atsuko Miyaji. Bidder scalable m+1-price auction with public verifiability. *2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, 2021.

[22] Kazumasa Omote and Atsuko Miyaji. A second-price sealed-bid auction with verifiable discriminant of p0-th root. In Matt Blaze, editor, *Financial Cryptography*, pages 57–71, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[23] Jonathan Levin. Auction theory, Oct 2004.

[24] Paul Milgrom and Robert J Weber. The value of information in a sealed-bid auction. *Journal of Mathematical Economics*, 10(1):105–114, 1982.

[25] Yili Hong, Chong (Alex) Wang, and Paul A. Pavlou. Comparing open and sealed bid auctions: Evidence from online labor markets. *Information Systems Research*, 27(1):49–69, 2016.

[26] Etherscan API. Etherscan export verified contracts.

[27] Raymond Cheng, Fan Zhang, Jernej Kos, Warren He, Nicholas Hynes, Noah Johnson, Ari Juels, Andrew Miller, and Dawn Song. Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, jun 2019.

[28] Rui Yuan, Yu-Bin Xia, Hai-Bo Chen, Bin-Yu Zang, and Jan Xie. Shadoweth: Private smart contract on public blockchain. *Journal of Computer Science and Technology*, 33(3):542–556, 2018.

[29] Thomas Kerber, Aggelos Kiayias, and Markulf Kohlweiss. Kachina - foundations of private smart contracts. Cryptology ePrint Archive, Paper 2020/543, 2020. https://eprint.iacr.org/2020/543.

[30] Guy Zyskind, Oz Nathan, and Alex Pentland. Enigma: Decentralized computation platform with guaranteed privacy, 2015.

[31] Aritra Banerjee, Michael Clear, and Hitesh Tewari. zkhawk: Practical private smart contracts from mpc-based hawk. In *2021 3rd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*, pages 245–248, 2021.

[32] Aritra Banerjee and Hitesh Tewari. Multiverse of hawkness: A universally-composable mpc-based hawk variant. Cryptology ePrint Archive, Paper 2022/421, 2022. https://eprint.iacr.org/2022/421.

[33] Samuel Steffen, Benjamin Bichsel, Roger Baumgartner, and Martin Vechev. Zeestar: Private smart contracts by homomorphic encryption and zero-knowledge proofs. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 179–197, 2022.

[34] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. Zether: Towards privacy in a smart contract world. Cryptology ePrint Archive, Paper 2019/191, 2019. https://eprint.iacr.org/2019/191.

[35] Ravital Solomon and Ghada Almashaqbeh. smartfhe: Privacy-preserving smart contracts from fully homomorphic encryption. Cryptology ePrint Archive, Paper 2021/133, 2021. https://eprint.iacr.org/2021/133.

[36] Yael Doweck and Ittay Eyal. Multi-party timed commitments, 2020.

[37] Kaili Wang, Qinchen Wang, and Dan Boneh. Erc-20r and erc-721r: Reversible transactions on ethereum, 2022.

[38] Chao Li and Balaji Palanisamy. Decentralized privacy-preserving timed execution in blockchain-based smart contract platforms, 2019.

[39] OriginStamp, 2022.

[40] Xiaohui Liu. Zero-overhead private timestamping in bitcoin, Apr 2022.

[41] Sean Bowe, Alessandro Chiesa, Matthew Green, Ian Miers, Pratyush Mishra, and Howard Wu. Zexe: Enabling decentralized private computation. Cryptology ePrint Archive, Paper 2018/962, 2018. https://eprint.iacr.org/2018/962.

[42] Universal blockchain explorer and search engine.

[43] Huixin Wu and Feng Wang. A survey of noninteractive zero knowledge proof system and its applications. *The Scientific World Journal*, 2014:1–7, 2014.

[44] Austin Mohr. A survey of zero-knowledge proofs with applications to cryptography, 01 2007.

[45] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. Cryptology ePrint Archive, Paper 2012/215, 2012. https://eprint.iacr.org/2012/215.

[46] Xiaoqiang Sun, F. Richard Yu, Peng Zhang, Zhiwei Sun, Weixin Xie, and Xiang Peng. A survey on zero-knowledge proof in blockchain. *IEEE Network*, 35(4):198–205, 2021.

[47] Silvio Simunic, Dalen Bernaca, and Kristijan Lenac. Verifiable computing applications in blockchain. *IEEE Access*, 9:156729–156745, 2021.

[48] Alex Ozdemir and Dan Boneh. Experimenting with collaborative zk-snarks: Zero-knowledge proofs for distributed secrets. Cryptology ePrint Archive, Paper 2021/1530, 2021. https://eprint.iacr.org/2021/1530.

[49] Juan Benet. Ipfs - content addressed, versioned, p2p file system, 2014.

[50] Ahmed Kosba, Charalampos Papamanthou, and Elaine Shi. xjsnark: A framework for efficient verifiable computation. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 944–961, 2018.

[51] Scipr-Lab. Scipr-lab/libsnark: C++ library for zksnarks.

[52] Ahmed Kosba. jsnark: A java library for zk-snark circuits.

[53] Jens Groth. On the size of pairing-based non-interactive arguments. Cryptology ePrint Archive, Paper 2016/260, 2016. https://eprint.iacr.org/2016/260.

[54] NomicFoundation. Nomicfoundation/hardhat: Hardhat is a development environment to compile, deploy, test, and debug your ethereum software. get solidity stack traces & console.log.

[55] ethereum.org.

[56] github.com/HarryR. Harryr/ethsnarks: A toolkit for viable zk-snarks on ethereum, web, mobile and desktop.

[57] Taechan Kim and Razvan Barbulescu. Extended tower number field sieve: A new complexity for the medium prime case. Cryptology ePrint Archive, Paper 2015/1027, 2015. https://eprint.iacr.org/2015/1027.

[58] Zcash. Understand the concrete security level of the bn_128 curve in libsnark · issue #714 · zcash/zcash.

[59] Ethereum. Eips/eip-170.md at master · ethereum/eips, Sep 2021.

[60] Christian Reitwiessner. Eip-196: Precompiled contracts for addition and scalar multiplication on the elliptic curve alt_bn128, Feb 2017.