Privacy Preserving Event based Transaction System in a Decentralized Environment

Rabimba Karanjai
rkaranjai@uh.edu
University of Houston
Texas, USA

Lei Xu
xuleimath@gmail.com
University of Texas Rio Grande Valley
Texas, USA

Zhimin Gao
mtion@msn.com
Auburn University at Montgomery
Alabama, USA

Lin Chen
Lin.Chen@ttu.edu
Texas Tech University
Texas, USA

Mudabbir Kaleem
mkaleem@uh.edu
University of Houston
Texas, USA

Weidong Shi
wshi3@uh.edu
University of Houston
Texas, USA

Abstract

In this paper, we present the design and implementation of a privacy preserving event based UTXO (Unspent Transaction Output) transaction system. Unlike the existing approaches that often depend on smart contracts where digital assets are first locked in a vault, and then released according to event triggers, the event based transaction system encodes event outcome as part of the UTXO note and safeguards event privacy by shielding it with zero-knowledge proof based protocols such that associations between UTXO notes and events are hidden from the validators. Without relying on any triggering mechanism, the proposed transaction system separates event processing from the transaction processing where confidential event based UTXO notes (event based UTXOs or conditional UTXOs) can be transferred freely with full privacy in an asynchronous manner, only with their asset values conditional to the linked event outcomes. The main advantage of such design is that it enables free trade of event based digital assets and prevents the assets from being locked. We implemented the proposed transaction system by extending the Zerocoin data model and protocols. The system is implemented and evaluated using xJsnark.

CCS Concepts

• **Software and its engineering** → **Distributed systems organizing principles**;

Keywords

UTXO, privacy, distributed ledger, zero-knowledge proof

ACM Reference Format:
Rabimba Karanjai, Lei Xu, Zhimin Gao, Lin Chen, Mudabbir Kaleem, and Weidong Shi. 2021. Privacy Preserving Event based Transaction System in a Decentralized Environment. In *22nd International Middleware Conference (Middleware '21), December 6–10, 2021, Virtual Event, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3464298.3493401

1 Introduction

In a traditional blockchain, all the transaction data is propagated and stored in every node. Nodes must be able to validate and execute each transaction, making it impossible to encrypt them. Since transactions are validated by replaying them, it will not be possible without access to the full data. In case of Bitcoin, if two people engage in a transaction, the public keys and amount transferred is visible to everyone on the network. And those public keys can be used to possibly de-anonymize the transaction and mark keys to a specific account or user by constructing and analyzing graphs of keys, their relationship and transactions [11, 35, 36].

The primary objective of this work is to enable a privacy-preserving event-based transaction system, specifically enhancing UTXO based model (e.g., UTXO based coins) with conditional values (conditional UTXOs or event-based UTXOs) and protecting event privacy by shielding the conditional UTXOs with zero-knowledge (zk) proof-based transaction protocols (e.g., Zcash [39]). In Unspent Transaction Outputs (UTXO) model, users spend their entire balance defined in a UTXO first, and then receive the unspent amount back. The amount sent to the recipient becomes a new UTXO in the recipient's address while the change output becomes a new UTXO in the sender's address that may be used in a future transaction. Even though UTXO based transactions in Bitcoin provide security features such as measures to prevent double-spending, but not privacy. Zcash uses zk-proofs thus generating a larger anonymity set to mitigate this problem and making the transactions private.

Although there exist designs connecting asset transactions with events [41] (such as issuing transactions based on event triggers), the proposed approach distinguishes from the prior efforts in several aspects. **Firstly**, the proposed event-based transaction system associates event conditions with UTXO based privacy coins, demonstrated using Zcash as a baseline model, which is named as conditional privacy coins or event-based privacy coins. **Secondly**, the links between events and conditional coins are kept confidential in order to achieve event condition privacy. Using zero-knowledge proof-based protocols, a conditional coin can be validated against its associated event outcome without disclosing which event or event outcome that it links to. **Thirdly**, conditional coins can be transferred and traded freely in the system before the associated event has occurred or its outcome has been declared (asynchronous to the event processing). There are no lock-up period of assets. Coins with conditions can be exchanged and transferred, just like regular

UTXO notes. It is the value of a conditional coin, not a transaction of the coin, that is conditional to the event outcome. This design has significant benefits over some existing or alternative synchronous designs that freeze assets in a vault or smart contract until the event result is published. **Fourthly**, the system decouples the event processing and the transaction processing. This mechanism facilitates the conditional coin processing and allows transactions of the conditional coins to be validated in an asynchronous manner from event handling. Such separation potentially can support rich and more complex event processing like event combination, event reasoning, and proposition logic over event conditions, to mention just a few.

To summarize, the main contributions of the paper are: (i) We proposed an event-based privacy-preserving transaction system where confidential event conditions are associated with shielded privacy coins (shielded conditional UTXOs) by extending the data model of privacy-based UTXOs. (ii) The proposed system and its protocol allow the conditional coins and the conditional digital assets to be transferred, exchanged, and traded freely in an asynchronous manner from the event processing logic. There is no asset lock-up or freeze of the conditional coins in a vault that a release is triggered in accordance with the event outcomes. It is the value of the coins or assets that is conditional to the event outcomes. (iii) We further developed optimization approaches applying K-anonymity and Merkle hash subtree verification in transactions to improve performance. (iv) We implemented the privacy-preserving event-based UTXO protocol in xJsnark using Zcash as a baseline model, and evaluated its performance.

2 Overview

Problem statement: To enable private and transferable conditional coins based on events, the system needs to meet the following requirements: (i) Privacy coins (e.g., use Zcash as a base model and initial target) are associated with event conditions. The conditional coins can be validated against event outcomes. However, links between the conditional coins and the events (both event definitions and event outcome announcements) should be hidden from the validators. (ii) The system should support complete transferability of the conditional coins where a conditional coin after its creation can be transferred unfettered by its current owner using the default privacy coin transaction protocol regardless if the event outcome has been declared or not. (iii) Event processing and transaction processing are decoupled. Event processing separates from the conditional coin transactions. Event processing includes registration of events and event outcome announcements/declarations. Updates to each part of the decoupled system are asynchronous from each other. (iv) All the coins in the system, regardless if they have event conditions attached or not, are indistinguishable. The system does not treat the privacy coins with event conditions differently from the coins without event conditions. Transactions involving privacy coins with event conditions are indistinguishable from the transactions of the coins without event conditions. When a transaction system satisfies this property, we call it supporting full privacy. Note that naive extension of the Zcash notes with event tags would not meet this requirement because notes can be distinguished by the event tags.

We assume in this work that all event conditions are binary. The design can be easily extended to non-binary event conditions. The system can support a wide range of application scenarios in decentralized finance, logistics, supply chains, prediction markets, risk management, insurance, to name just a few. For instance, events can be defined such as "Bitcoin price increases by 5%", "Interests rate is lowered by more than 2% in the coming month", "Goods shipped with tracking number 1Z54F78A0450293517 is delivered", "Australia qualifies for semi-final of world cup 2023", "Gold index rose more than 1%".

System overview: To meet these requirements, firstly, we adopt a system where event processing decouples from the transaction processing. There are separate logical chains of records for registering event definitions, announcements of the event outcomes, and conditional privacy coin transactions [1]. In this paper, we assume that there is a global distributed ledger for supporting all the three types of data records. In addition, the system defines constant events and constant event outcomes in the genesis block.

Transaction examples: Figure 1 provides an example of association between event definitions, event outcome announcements, and event based UTXO notes. It shows nine UTXO notes (named cc1 to cc9) and four transactions (named tx1 to tx4). For each transaction, input notes and output notes are enclosed in a colored bounding box. The coins in front of the arrows are input coins and the coins that the arrows point to are output coins. Each type of data has its own logical chain and the corresponding Merkle hash tree. All the event definitions and event outcome announcements are in public. They are validated before appended to the ledger. In this work, we assume that event outcomes are declared by the same parties who register the events to the ledger (verified through a secure digital signature scheme). Verification of the event outcome itself (e.g., status of a tracked shipment) is a separate concern, which could be supported through various established approaches such as oracle service providers, event validators or zero-knowledge proof based claim verification scheme (e.g., [43]). We assume that event outcomes are verified as part of the event processing before they are appended to the shared ledger.

All privacy coins in the system are conditional coins. In the example, one can observe that all the coins (cc1 to cc9) are paired with an event definition. A privacy coin without a real condition is configured to use one of the default constant events as its condition, for instance cc1. This way, all the transactions involving the conditional coins in the system (payments using the conditional coins or transferring of the conditional coins) follow identical validation steps in zero-knowledge protocol when deciding whether the transactions should be accepted by the participating distributed ledger nodes.

When a privacy coin with a default constant event as its condition is transferred or spent, the output coins may maintain the same constant event as conditions. To associate a privacy coin note with an event outcome, an input privacy coin with a satisfied event condition can be split into two output coins with outcomes of a new event as conditions. The two output coins must have opposite event

[1]These separate chains of records (event definitions, event announcements, conditional coin transactions) can be implemented either as one distributed ledger or as separate ledgers. We leave such option as specific implementation issue independent from the data models and conditional coin protocol design.

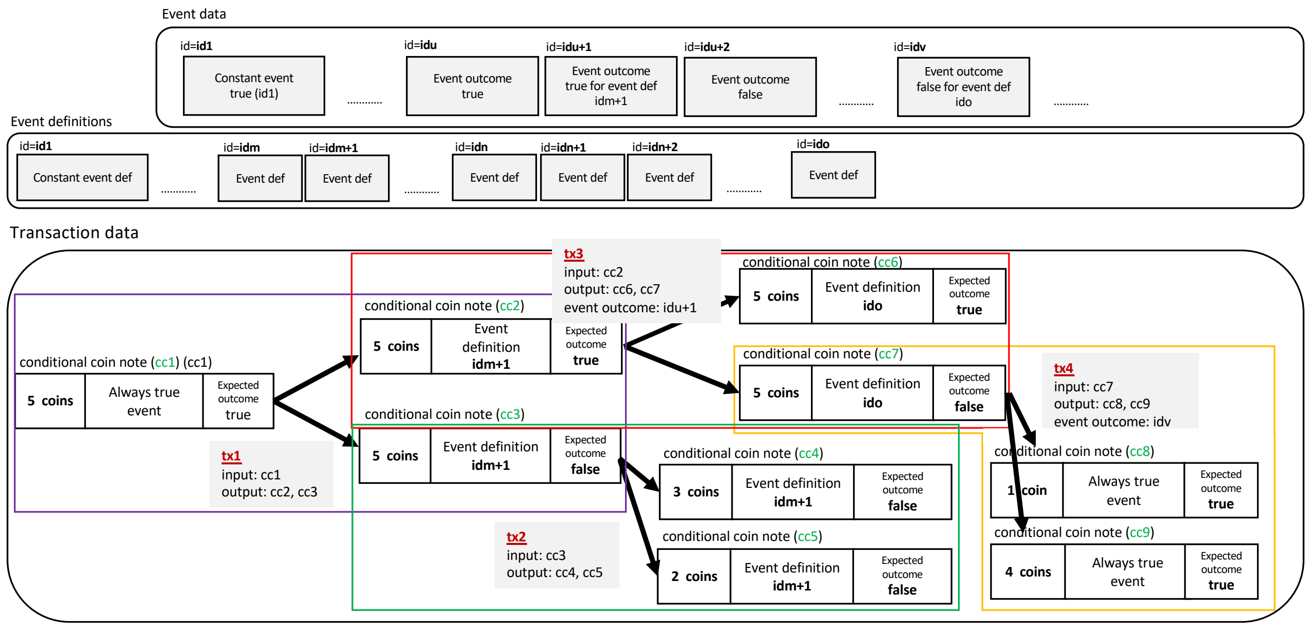

Figure 1: Illustration of event based UTXO (conditional coins) transactions.

outcomes as their conditions, as illustrated by the two transactions tx1 and tx3 in Figure 1. Each output coin has the same total value as the input.

The conditional coins can be transferred freely as demonstrated by tx2 in Figure 1. In tx2, conditional coin cc3 is split into two output coins, cc4 and cc5, which can be transferred to two different recipients or the same recipient. To remove the condition, the coin spent needs to show a matched event outcome, see tx3 and tx4 in Figure 1. Both transactions have matched event outcome. Validation is done using zero-knowledge proof without leaking any information of the associated event definition and event outcome. After the condition is removed, a conditional coin output can either have a different event condition attached, or switch back to one of the default events as its condition.

Under this framework, it is possible to support complex event processing and event reasoning. A calculus system of events can be defined; and new complex events can be created based on the existing events. In addition, event processing can be augmented with smart contract support to enable diverse means for validating event outcomes.

Applications: As a transaction system, it can enable and facilitate a wide range of applications including but not limited to, transferable confidential assets with valuation based on the event outcomes, bill of exchange based on digital assets, future contracts, prediction market, and use case areas like logistics, FinTech, insurance (such as enable secondary market), etc.

Challenges: To realize an event based transaction system with full privacy support, one has to solve the following challenges: *design of event based privacy coin fully compatible with the UTXO model, asynchronous transaction model where transfers of the event tagged conditional coins are asynchronous to the event processing, extension to the transaction protocols with full privacy support including event privacy, optimization of the implemented protocols to reduce overhead like proving time and proving circuit complexity.* To remain focused, specific questions like choosing consensus mechanism (e.g., Proof-of-Stake, PBFT, Proof-of-Work) are out of the scope of this paper. The protocols are designed to be agnostic to any consensus mechanism.

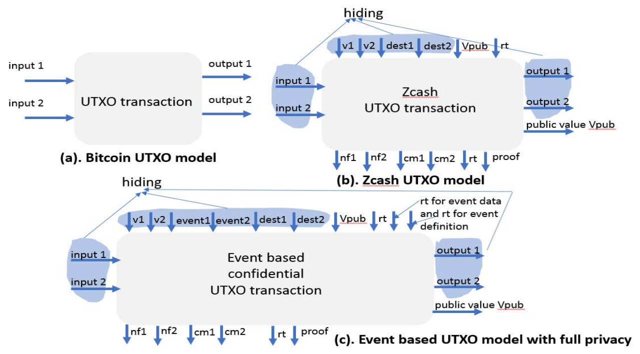

Figure 2: Illustration of different UTXO models. Data under shaded areas are hidden from the public.

3 Background

Transactions are the fundamental building blocks in distributed ledgers or blockchains. Transactions represent state changes in a distributed ledger.

UTXO model: The UTXO model is a common design adopted by many blockchain based transaction systems, for instance Bitcoin. In the UTXO model, tracking digital assets does not use accounts or balances. Instead, individual coins (UTXOs also called UTXO notes) are transferred between users in a way similar to physical coins or cash. Each node in the system keeps track of status of UTXOs, which includes detection of double spending.

When a transaction happens, it comprises of transaction version, which tells a node the set of rules to be used to verify it, followed by lock-time which specifies whether a transaction can be included in the blockchain right away or after some delay. Before a lock-time takes place, a transaction output is locked. In a UTXO transaction, input provide a key to unlock previously locked output. UTXO notes are data records that contain an amount indicating the quantity of value to be transferred, a locking script and other metadata. Locking scripts within a transaction output encode the conditions that must be satisfied in order for the amount to be spent. Transaction inputs contain a transaction hash, output index, unlocking script, and other metadata. The transaction hash is a hash of the transaction containing the transaction output from where the value is to be drawn. Every transaction contains a pointer and an unlocking key. The pointer points to a previous transaction output which the key is used to unlock. The unlocking script contains necessary information to the locking script of the referenced UTXO, which can be validated by other nodes in the blockchain network. Every time an unlock happens, it is marked into the blockchain as "spent".

Every node in a blockchain network keeps track of the available UTXOs in a data structure called UTXO pool. When a UTXO is matched with a transaction input, it is considered as spent, and therefore is removed from the UTXO pool. This means that a UTXO can be only spent once. Figure 2 demonstrates and compares three UTXO models: the Bitcoin UTXO transaction, the Zcash's shielded UTXO transaction, and our event based UTXO transaction with full privacy protection. The latter two models are discussed at the end of Section 3 and the succeeding sections.

Oracle: Blockchains and smart contracts cannot access off-chain data. However, for many contractual agreements, it is vital to have relevant information from outside to execute the agreement. This creates the need for data feeds designed to bring external data into a distributed ledger system. These data feeds are known as oracles. Oracles are vital within the blockchain ecosystem because they broaden the scope in which a transaction system or smart contracts can operate. Without blockchain oracles, the systems would have very limited use as they would only have access to data from within their networks.

Oracles can act as trusted data sources for on-chain transactions. They can be applied to both smart contract and UTXO based systems. There are diverse design patterns for oracles that differ in terms of the trust model, decentralization, interaction process, efficiency, cost, and etc (e.g., [1]). Centralized oracle model relies on a single trusted intermediary as a bridge for external data source. This approach is very efficient and cost effective. However, centralized oracles bring the concern about trustworthiness and validity of the data that depend on a single trusted source. To resolve such concern, decentralized oracle is developed. Decentralized oracle network is a group of independent blockchain oracles that provide data to a blockchain. Oracle node in the decentralized oracle network independently retrieves data from an off-chain source and brings it on-chain. these are then aggregated so the system can come to a deterministic value of truth for that data point based on consensus [18]. Decentralized oracle model addresses the trust issue of centralized oracles at the cost of efficiency (e.g., higher latency and cost). While decentralized oracles aim to achieve trustlessness, it is important to note that decentralized oracles do not completely eliminate trust, but rather distribute it between many participants.

This paper does not focus on research of oracle design. It assumes the existence of oracle service used for the event processing and validation.

Zero-knowledge proof based UTXO model: In the default UTXO model, all the transactions are in public, which brings privacy concerns. To resolve such concern, zero-knowledge based transaction protocols such as Zcash/Zcoin are developed (as demonstrated in Figure 2.(b)). A lot of the work that went to implement a private blockchain relies on zero knowledge proofs, specially zk-SNARK [3, 33] , which enables zero knowledge proof in Zcash. A zero-knowledge proof of knowledge is a protocol in which a prover can convince a verifier that some statement holds without revealing any information. Roughly speaking, a zero-knowledge proof involves two parties, the *prover* and the *verifier*. For a statement, the *prover* can generate a proof to convince the *verifier* of the correctness of the statement. In this process, the *verifier* cannot learn anything except the fact that the statement is true (zero-knowledge feature). Zero-knowledge proofs can be either interactive or non-interactive. Interactive zero-knowledge proofs [15, 16] require the *prover* to communicate with the *verifier* over multiple rounds to finish the proof. Non-interactive zero-knowledge proofs (NIZK) [4, 13, 17] do not require multiple rounds of interactions between the *prover* and the *verifier* and are more suitable for scenarios where it is difficult for these parties to be online at the same time.

When a UTXO note is spent, the spender (or payer) needs to create a zero-knowledge proof that there is a corresponding note commitment (commit) in the pool of commitments without disclosing which one it is. The proof also shows that the transaction is consistent and that the user knows all the secret keys, without revealing any additional information. In Zcash, such protected transactions are called shielded transactions.

4 Data Models

The data models are based on extending the notations of Zcash [19, 28]. In this paper, for simplification, we apply zk-SNARK and the original Zcash design [19, 28] to achieve an exemplary implementation. However, it is worthwhile pointing out that it is plausible to realize the described data models and protocol using alternative back-end zero-knowledge proving systems such as zk-STARK [2], bulletproof [7]. Discussion of leveraging different back-end zero-knowledge proving systems is orthogonal to the scope of this work, which focuses primarily on the front-end protocol design of a privacy preserving transaction system for the event based UTXOs.

Conditional privacy coin ledger: There is a distributed ledger, L_{CPC}, which records a sequence of transactions in append-only mode. The ledger could be implemented as a blockchain where transactions are recorded as blockchain blocks. The ledger supports multiple types of transactions designed for event registration and declaration as well as transferring of the conditional coins. The ledger comprises ordered blocks created from a genesis block under a consensus mechanism. Each block has a block height. Further, we assume that blocks are generated with a relatively constant speed. Details of the consensus mechanism and block generation could either be based on or follow the Zcash design. Although our data

Table 1: Data definitions

	Event definition attributes
eID	64-bit unique identifier for event definition
bt	64-bit time threshold of event outcome (block height)
repeat	event definition with repeated outcome announcements
ba	event outcome announced before or after the time limit
pk_{sig}	public key of a public/private signature key pair (pk_{sig}, sk_{sig}) for event outcome announcement
	Event announcement attributes
eaID	64-bit unique identifier for event outcome announcement
eID	64-bit identifier for the associated event definition
bh	64-bit time stamp (block height)
v	event outcome value
	Shielded event based UTXO attributes - extended Zcash coin definition
a_{pk}	public key of payment address pair (a_{pk}, a_{sk}) where a_{sk} is the spending key
v	value of coin
eID	64-bit identifier for the associated event definition
cond	expected event outcome
ρ	used to compute nullifier (disclosed to the public after spending)
γ	trapdoor
cm	note commitment (comm or commit)

models and protocols are based on extending the Zcash specification, the design can be adapted to any blockchain based systems, for instance substituting the underlying consensus mechanism with Proof-of-Stake (PoS) or PBFT based design.

There are three main types of data records: (i) records for the event registration and definition; (ii) records for the event outcome declaration; and (iii) records for the conditional coin transaction and transferring.

The data models are summarized in Table 1. UTXO note format of the conditional coins is based on extension over the Zcash note format.

Event definition: The system defines events as tuples of attributes as shown in Table 1. After an event definition is registered to the ledger, there could be one or multiple event declarations or announcements associated with it (based on the single bit attribute *repeat* - *true* or *false*). Each event definition is uniquely identified with a 64-bit value *eID*. Attribute *bt* is a block height value that specifies when value of the event (outcome of the event) should be declared or announced to the system. When *ba* is *true*, event outcome should be announced before the block height set by *bt*. Otherwise when *ba* is *false*, event outcome should be announced after block *bt*.

To allow only the authorized parties to declare event outcomes, for each registered event definition, there is a public/private signature key pair (pk_{sig}, sk_{sig}). Public key pk_{sig} is disclosed during registration. When outcome of the event is declared sometime later, private key sk_{sig} will be used to sign the event transaction that announces the event outcome. For repetitive event, a new signature key pair will be created each time there is an event outcome declaration.

In addition, there are default event definitions that serve as constant events (“always *true*” and “always *false*”), $edef^{dft}_{true}$ and $edef^{dft}_{false}$. These constant events are defined in the genesis block. To prevent denial-of-service, a fee is charged for event registration.

Event outcome announcement: For a registered event definition, its outcome will be declared with an event outcome transaction. Each outcome announcement includes the attributes described in Table 1. Attribute *eID* specifies the associated event ID (as a key linking to the corresponding event definition). Attribute *eaID* is a 64-bit value that uniquely identifies an event outcome declaration. Attribute *bh* serves as a timestamp value in block height. Event outcome is encoded in attribute v. In this work, we assume binary event outcome. Note that complex events can be converted to binary events. The system can be extended to support more complex event outcome scenarios. For the default constant events, their event outcomes are defined in the genesis block as ea^{dft}_{false} for “always false” and ea^{dft}_{true} for “always true”.

Transaction address and conditional UTXO note format: We use the same Zcash design of payment address pair (a_{sk}, a_{pk}) where a_{sk} is used as spending key. For receiving a shielded payment, a user needs to scan ledger L_{CPC} using (a_{pk}, sk_{enc}). The algorithm is similar to the one described in Zcash blockchain scanning. There is a v_{pub}, similar to the public value in Zcash. Coin values embedded in conditional notes can be transferred to v_{pub} and vice versa, see Figure 2.(c)).

An event based UTXO note, n, is similar to the Zcash note with extensions to support pairing of the note with an event definition and event outcome. Note attributes including a_{pk}, ρ, γ, and v are the same as defined in Zcash. For each note, attribute *eID* associates the note with an event definition where *eID* serves as the key. Attribute *cond* specifies the anticipated event outcome. When declared event outcome matches with the expected event outcome encoded in a conditional note, the note will maintain its coin value v. Otherwise, the note will lose its value.

An event based note can be spent or transferred without waiting for announcement of the anticipated event outcome. This characteristic, transferability of the conditional coins/notes before event occurrence, makes our system distinguishing from other designs that rely on event triggers. Note that even a conditional coin note loses its value, it can still be transferred.

When a note is spent, a nullifier value, nf, will be created using attribute ρ as input where nf is determined by $\mathsf{PRF}^{nf}_{a_{sk}}(\rho)$. *PRF* is a pseudo-random function that can be implemented using *SHA2*. The decentralized system enforces that nullifiers must be unique in order to prevent double-spending.

Multiple Merkle trees: L_{CPC} uses incremental Merkle trees of fixed depth for event definitions, event outcome announcements, and note commitments. In this work, for simplicity, we assume that there are separate Merkle trees, M^{edef}, M^{ea}, and M^{tx} for each data record type. Note that Zcash applies only one Merkle tree for note commitments. It is plausible to have one unified Merkle tree for all the data records. Each Merkle tree has its own root, denoted as rt^{edef}, rt^{ea}, and rt^{tx} respectively. In a Merkle tree, we represent the location of a data record as (*pos*, *path*) where *pos* is the leaf node position in the tree, and *path* is the path of Merkle hash computation from the leaf record to the root.

Public parameters: In the case of the experiments in this work, the system uses the same set of public parameters *pp* in the Zcash design. They are generated either by a trusted party at the beginning or through a Multi-Party Computation ceremony [5, 6]. If the system is implemented over a proving system that does not require trusted setup, the public parameters can be based on common public strings appropriate for the proving system.

Note that in this work, we restrict privacy preserving event based UTXO transactions to the cases of two input notes and two output notes. Transactions involving more than two input notes or two output notes can be reduced to a series of transactions, each only

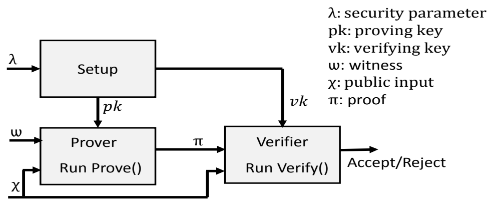

Figure 3: Diagram of zero-knowledge proving system.

involving two notes as input and output. Similar to the Zcash design, an input note or an output note can be a dummy note (without associated note commitment). In the case of dummy note, the asset value is zero.

5 Protocol Design for Privacy Preserving Event based UTXOs

5.1 Preliminary

The protocol extends the design of zero-knowledge based privacy transaction system with Zcash as the baseline model. Privacy oriented blockchains apply zero-knowledge proving system as the underlying building block to protect privacy. A zero-knowledge proving system [19] defines the parameters and process illustrated in Figure 3.

A zero-knowledge proving system needs to satisfy the following security requirements: completeness, succinctness, and proof-of-knowledge. (i) *Completeness:* for security parameter λ, a $\mathbb{F}$, arithmetic circuit C [34], and any $(\chi, \omega) \in \mathcal{R}$ (satisfying inputs), honest prover can convince the verifier with probability $1 - negl(\lambda)$. (ii) *Succinctness:* honestly generated proof π has $O_n(1)$ bits and Verify(vk, χ, π) runs in the $O_\lambda(|\chi|)$. (iii) *Proof-of-knowledge:* If the verifier accepts a proof output by a bounded prover, then the prover knows witness for the given statement.

5.2 Transaction Algorithms

For completeness, this subsection lists the main algorithms. The main extensions are related to the JoinSplit transaction. JoinSplit is a new structure added to the Bitcoin transaction format. A JoinSplit contains three essential fields: the number of coins entering the shielded pool, the number of coins exiting the shielded pool, and a field holding a zero-knowledge proof attesting to the legitimacy of the transaction, see Figure 2 that compares three transaction models, the Bitcoin UTXO, the Zcash UTXO, and our extension for confidential event based UTXO. JoinSplit is extended to support associating a shielded UTXO note with an event outcome. To highlight the changes, in algorithm 1, 2, and 3, data and steps unique to our system are marked in blue font. Note that both Zcash and the event based extension with full privacy follow the same process flow described in Figure 3 for generating proofs by a node who issues transactions and verifying proofs by all the other nodes in the systems.

Setup$_{CPC}$: Follows the same Zcash algorithm for constructing public parameters. $Setup_{CPC}$ takes 1^λ as security parameter and outputs public parameters pp_{CPC} that includes: $pk_{JoinSplit}$, $vk_{JoinSplit}$, pp_{enc}, pp_{sig}. Parameter $pk_{JoinSplit}$ and $vk_{JoinSplit}$ are a pair of proving and verifying keys for JoinSplit transactions. pp_{enc} is an encryption key and pp_{sig} is a signature key.

CreateAddr$_{CPC}$: It takes public parameters pp_{CPC} as input and creates a pair of transmission key (a_{pk}, pk_{enc}) and receiving key (a_{pk}, sk_{enc}) where a note sent to a recipient is encrypted using pk_{enc} and it is retrieved by the recipient from the ledger using sk_{enc}. Similar to the Zcash design, the algorithm creates sk_{enc} and pk_{enc} key pair using Curve25519 key agreement. Interested readers can refer to the Zcash documentation for details.

Mint$_{CPC}$: It takes public parameters pp_{CPC}, public address pair (a_{pk}, pk_{enc}), π, value v where $v \in (0, ..., v_{max})$, a default constant event definition. It appends note n to the ledger L_{CPC}, or output $\perp$ (reject). This operation is used to mint notes based on the public value and put the notes on L_{CPC}. The algorithm is similar as defined in Zcash with the main difference that the newly minted UTXOs encode the default event conditions (e.g., "always true").

JoinSplit$_{CPC}$: JoinSplit$_{CPC}$ takes a set of input values and creates two output notes, n_1^{new} and n_2^{new}, and a transaction $tx_{JoinSplit}$. Algorithm 1 lists pseudo-code adapted from the Zcash specification. With such transactions, conditional coin notes can be created by spending the existing notes and transferred regardless whether the event outcome has been declared. Data introduced in our system is highlighted in blue font.

Algorithm 1: JoinSplit algorithm for shielded event based UTXOs (conditional coins with privacy).

```
Input  : Public parameters $pp_{CPC}$, ledger $L_{CPC}$ (including Merkle tree $M^{tx}$
         and root $rt^{tx}$, Merkle tree $M^{edef}$ and root $rt^{edef}$, Merkle tree $M^{ea}$
         and root $rt^{ea}$), input note $n_1^{old}$ and $n_2^{old}$, input note spending keys
         $a_{sk,1}^{old}$ and $a_{sk,2}^{old}$, event outcomes associated with the input note
         $ea_1^{old}$ and $ea_2^{old}$, event definition paired with the new notes
         $edef^{new}$, output addresses $a_{pk,1}^{new}$ and $a_{pk,2}^{new}$, public value $v_{pub}^{old}$,
         block height $block_n$
Output : A transaction $tx_{join\_split}$ with two output conditional coin notes
1  For each input note, compute nullifier $nf_i^{old} = \mathsf{PRF}^{nf\,old}_{a_{sk,i}^{old}}(n_i^{old}.\rho) =$
   $SHA256(a_{sk,i}^{old} \| n_i^{old}.\rho)$
2  Create signature key pair $(pk_{sig}, sk_{sig}) = \kappa_{sig}(pp_{sig})$ where $\kappa_{sig}$ generates a
   key pair for signature signing
3  Compute $h_{sig} = \mathsf{CRH}(nf_1^{old}, nf_2^{old}, pk_{sig})$ where CRH is a collision resistant
   hash function
4  Set $\rho_i^{new} = \mathsf{PRF}^{\rho}_{\varphi}(i, h_{sig}) = SHA256(i \| \varphi \| h_{sig})$
5  For each new note, sample a random $\gamma_i^{new}$
6  Set new conditional note and its attributes as $n_i^{new} = (a_{pk,i}^{new}, v_i^{new}, edef^{new},$
   $cond_i^{new}, \rho_i^{new}, \gamma_i^{new})$
7  Compute new note commitment $cm_i^{new} = NoteCommit(n_i^{new})$ where
   NoteCommit is based on SHA256
8  Compute old note spending signature $h_i = \mathsf{PRF}^{pk}_{a_{sk,i}^{old}}(i \| h_{sig})$
9  Encrypt new note as $N_i^{enc,new} = \mathcal{E}_{enc}(pk_{enc,i}^{new}, n_i^{new})$
10 Set public input $\chi = (rt^{tx}, rt^{edef}, rt^{ea}, nf_1^{old}, nf_2^{old}, cm_1^{new}, cm_2^{new}, v_{pub}^{old}, v_{pub}^{new},$
   $block_n, h_{sig}, h_1, h_2)$
11 Set witness $\omega = (n_1^{old}, n_2^{old}, a_{sk,1}^{old}, a_{sk,2}^{old}, \varphi, dummy_1^{old}, dummy_2^{old}, pos_{n,1}^{old},$
   $path_{n,1}^{old}, pos_{n,2}^{old}, path_{n,2}^{old}, ea_1^{old}, ea_2^{old}, pos_{ea,1}^{old}, path_{ea,1}^{old}, pos_{ea,2}^{old}, path_{ea,2}^{old},$
   $n_1^{new}, n_2^{new}, edef^{new}, pos_{edef}^{new}, path_{edef}^{new})$ - (see Table 2)
12 Compute proof $\pi_{JoinSplit} = Prove(pk_{JoinSplit}, \chi, \omega)$ - see Algorithm 2
13 Set transaction message $m = (\chi, \pi_{JoinSplit}, N_1^{enc,new}, N_2^{enc,new})$
14 Set message signature $\delta = S_{sig}(sk_{sig}, m)$
15 Set $tx_{join\_split} = (rt^{tx}, nf_1^{old}, nf_2^{old}, cm_1^{new}, cm_2^{new}, v_{pub}^{new}, *)$ where $* = (pk_{sig}, h_1,$
   $h_2, \pi_{JoinSplit}, N_1^{enc,new}, N_2^{enc,new}, \delta)$
```

When generating $\pi_{JoinSplit}$ (proof), the proving circuit verifies a set of constraints for the generated transaction, which is applied to

Table 2: Data fields defined in ω

$n^{old}_{1..N}$	old conditional coin notes, N=1 or 2
$a^{old}_{sk,1..N}$	old note spending key, N=1 or 2
$pos^{old}_{n,1..N}$	Merkle tree (M^{tx}) positions of the old note commitments, N=1 or 2
$path^{old}_{n,1..N}$	Merkle tree (M^{tx}) paths of the old note commitments, N=1 or 2
φ	random seed
$ea^{old}_{1..N}$	event outcome announcements associated with the old conditional coin notes, N=1 or 2
$pos^{old}_{ea,1..N}$	Merkle tree (M^{ea}) positions of the event outcome announcements associated with the old notes, N=1 or 2
$path^{old}_{ea,1..N}$	Merkle tree (M^{ea}) paths of the event outcome announcements associated with the old notes, N=1 or 2
$dummy^{old}_{1..N}$	one of the old notes is a dummy note or not, N=1 or 2
$n^{new}_{1..N}$	new conditional coin notes, N=1 or 2
$edef^{new}$	event definition associated with the new conditional coin notes
pos^{new}_{edef}	Merkle tree (M^{edef}) position of the event definition associated with the new note
$path^{new}_{edef}$	Merkle tree (M^{edef}) path of the event definition associated with the new note

generate a proof to be checked by a verifier. The pseudo-code of the proving circuit is listed in Algorithm 2. Definition of witness can be found in Table 2. Note that the algorithm verifies additional constraints related to events not present in Zcash, all steps highlighted in blue font. It verifies validity of the event definition associated with both the input note and the output note. Moreover, it verifies validity of the event outcome associated with the input note depending on the spending scenario. Different spending scenarios like illustrated by the four transaction examples in Figure 1 are verified as sub-circuits. Specifically, line 13 - 15 verifies transaction that transfers conditional coins. Line 16 - 17 verifies the default scenario like a regular Zcash transaction. All the transactions without event dependency can be translated to this scenario. Line 18 - 20 verifies the spending scenarios that remove the old event tag and attach a new event tag to the output notes. Line 21 - 22 verifies the spending scenarios that remove the old event tag and use the default event outcome - “always true” for output notes.

Receive$_{CPC}$: Given public parameters pp_{CPC}, Merkle trees and their roots, recipient key pair (a_{pk}, sk_{enc}), and the ledger L_{CPC}, it outputs received note n^{new}, or output $\perp$. The algorithm scans the ledger and outputs received note for each JoinSplit transaction.

Verify$_{CPC}$: It takes a set of inputs as described in Algorithm 3. It appends $tx_{JoinSplit}$, n^{new}_1 and n^{new}_2 to the ledger, or output $\perp$.

5.3 Event Transactions

Algorithm 4 creates a new transaction tx_{ea} for an event outcome given public parameters pp_{CPC}, ledger L_{CPC}, and the corresponding event definition $edef$.

After tx_{ea} is received by the blockchain nodes, it will be validated. If the event outcome can be accepted, it will be appended to the ledger L_{CPC} including update of Merkle tree M^{ea} and its root rt^{ea}.

6 Security Analysis

The event based transaction protocol is based on extending the data models and the transaction design of the original Zcash protocol. Since the algorithms are extensions within the Zcash framework, most of the security properties that are proven in the Zcash design and related publications still hold for the event based privacy notes [28][23][14][21]. This means that we can focus on the properties unique to the conditional coin protocol.

Algorithm 2: Prove algorithm.

```
Input  : $pk_{JoinSplit}, \chi, \omega(see Table\ 2)$
Output : $\pi_{JoinSplit}$
1  Verify common constraints below:
2  Merkle path validity for old notes: $(pos^{old}_{n,i}, path^{old}_{n,i})$ valid tree path from NoteCommit
     $(n^{old}_i)$ to the Merkle tree root $rt^{tx}$
3  Nullifier integrity: $nf^{old}_i = \mathsf{PRF}^{nf\ old}_{a^{old}_{sk,i}}(n^{old}_i.\rho)$
4  Spending key validity: $a^{old}_{pk_i} = \mathsf{PRF}^{addr}_{a^{old}_{sk,i}}(0)$
5  Uniqueness of $\rho^{new}_i$: $\rho^{new}_i = \mathsf{PRF}^{\rho}_{\varphi}(i, h_{sig})$
6  Note commit validity: $cm^{new}_{n,i} = NoteCommit(n^{new}_i)$
7  Event definition validity in new notes: $n^{new}_1.eID = n^{new}_2.eID = edef^{new}.eID$
8  Merkle path validity for the event definition associated with the new UTXO notes: $(pos^{new}_{edef},$
     $path^{new}_{edef})$ valid tree path from $edef^{new}$ to the Merkle tree root $rt^{edef}$
9  When $(n^{new}_i.eID \neq n^{old}_i.eID) \vee (n^{new}_i.eID = n^{old}_i.eID \wedge n^{new}_i.eID = (edef^{dft}_{true}.eID$
     $\vee\ edef^{dft}_{false}.eID))$:
10   Event condition validity: $n^{old}_i.cond = ea^{old}_i.v$
11   Merkle path validity for the event announcements associated with the old UTXO notes:
       $(pos^{old}_{ea,i}, path^{old}_{ea,i})$ valid tree path from $ea^{old}_i$ to the Merkle tree root $rt^{ea}$
12 Sub-circuits to validate constraints for different event based UTXO transaction scenarios:
13 Case: $(n^{new}_i.eID = n^{old}_i.eID) \wedge (n^{new}_i.eID \neq (edef^{dft}_{true}.eID \vee edef^{dft}_{false}.eID))$ - tx2 in
     Figure 1 as an example
14   Value validity: $v^{old}_1 + v^{old}_2 = v^{new}_1 + v^{new}_2$
15   Event condition validity: $n^{new}_1.cond = n^{new}_2.cond = n^{old}_i.cond$ where $i$ is either 1 and/or
       2 for non dummy input note
16 Case: $(n^{new}_i.eID = n^{old}_i.eID) \wedge (n^{new}_i.eID = (edef^{dft}_{true}.eID \vee edef^{dft}_{false}.eID))$ - default
     baseline case
17   Value validity: $v^{old}_1 + v^{old}_2 + v^{old}_{pub} = v^{new}_1 + v^{new}_2 + v^{new}_{pub}$
18 Case: $(n^{new}_i.eID \neq n^{old}_i.eID) \wedge (n^{new}_i.eID \neq (edef^{dft}_{true}.eID \vee edef^{dft}_{false}.eID)$ ) - tx3 or tx1
     in Figure 1 as examples
19   Value validity: $(v^{new}_1 = v^{new}_2) \wedge (v^{old}_1 + v^{old}_2 + v^{old}_{pub} = v^{new}_1 + v^{new}_{pub})$
20   New note event condition validity: $(n^{new}_1.cond = true \wedge n^{new}_2.cond = false) \vee$
       $(n^{new}_2.cond = true \wedge n^{new}_1.cond = false)$
21 Case: $(n^{new}_i.eID \neq n^{old}_i.eID) \wedge (n^{new}_i.eID = (edef^{dft}_{true}.eID \vee edef^{dft}_{false}.eID)$ ) - tx4 in
     Figure 1 as an example
22   Value validity: $v^{old}_1 + v^{old}_2 + v^{old}_{pub} = v^{new}_1 + v^{new}_2 + v^{new}_{pub}$
```

Algorithm 3: Verify algorithm.

```
Input  : Public parameters $pp_{CPC}$, ledger $L_{CPC}$, public input $\chi$, a JoinSplit transaction
         $tx_{join\_split}$, two notes $n^{new}_1$ and $n^{new}_2$
Output : accept or reject
1  Parse $tx_{join\_split}$
2  Set $b1 \leftarrow Verify(vk_{JoinSplit}, \chi, \pi_{JoinSplit})$
3  Set $b2 \leftarrow v_{sig}(pk_{sig}, m, \delta)$ where $m$ is transaction message and $\delta$ is message signature
     defined in Algorithm 1
4  Output $\perp$ if any of the following is true:
       • $b1 \wedge b2$ is false
       • $nf^{old}_1$ or $nf^{old}_2$ appears on $L_{CPC}$ - detect double spending
       • $nf^{old}_1 = nf^{old}_2$
       • Merkle root $rt^{tx}$ not on $L_{CPC}$
       • Merkle root $rt^{edef}$ not on $L_{CPC}$
       • Merkle root $rt^{ea}$ not on $L_{CPC}$
       • $h_{sig}$ does not match with $\mathsf{CRH}(nf^{old}_1, nf^{old}_2, pk_{sig})$
```

We assume that the underlying blockchain system to support distributed consensus and network communications is secure and reliable, which is outside the scope of this work. We further assume that measures are taken to prevent attacks that target protocol layers outside focus of this work such as eclipse attack, 51% attack, denial-of-service at network level, side-channel exploits for instance using time delay in transactions as side-channel information, privacy leakage through network traffic patterns, attack to DNS, hard

Algorithm 4: Algorithm for event announcement transaction.

```
Input  : Public parameters $pp_{CPC}$, ledger $L_{CPC}$, input event definition $edef$, event
         outcome value $v$, $sk_{sig}$, block height $block_n$
Output : Event outcome announcements associated with the event definition $edef$
1 Sample a new eaID
2 Set $ea = (edef.eID, eaID, v, block_n)$
3 When $edef.repeat$ is true
4 Create a new signature key pair $(pk_{sig}^{new}, sk_{sig}^{new}) = \kappa_{sig}(pp_{sig})$ where $\kappa_{sig}$ generates a key
  pair for signature signing
5 Sample a new $eID^{new}$
6 Set $edef^{new} = (eID^{new}, edef.repeat, bt^{new}, edef.ba, pk_{sig}^{new})$
7 Set transaction message $m = (ea, edef^{new})$
8 Set message signature $\delta = S_{sig}(sk_{sig}, m)$
9 Set $tx_{ea} = (m, \delta)$
```

forks, quantum attack, and etc. In addition, the system will ensure that event registration and event outcome declaration are properly secured. Event outcome is validated by the participating nodes of the decentralized ledger. Only the entity that registers an event can declare event outcome in an event outcome transaction. The transaction is protected by a secure SUF-CMA signature scheme (SUF-CMA stands for Strong Existential Unforgeability under Chosen Message Attack), which prevents forgery of event outcome transactions. Moreover, it is assumed that there is no denial-of-service for event outcome (event outcome will be declared within a bounded time [2]); and for each registered event, the system accepts only one result [3]. Last but not the least, we consider that the underlying zero-knowledge proving system is secure.

The original Zcash privacy coin defines security as: (i) *Ledger indistinguishability.* ledger reveals no information to the adversary $\mathcal{A}$ beyond publicly disclosed information; (ii) *Transaction non-malleability.* No bounded adversary $\mathcal{A}$ can alter any of the data stored within a valid transaction. It prevents the adversary from modifying others' transactions before they are added to the ledger, and (iii) *Balance.* No bounded adversary $\mathcal{A}$ can own more notes than what he/she minted or received via transactions from others.

In the case of conditional coins, transaction indistinguishability is extended to cover the requirement that no bounded adversary $\mathcal{A}$ can distinguish transactions with event conditions and transactions without event conditions. It is apparent that this property is satisfied because, in the described conditional coin protocol, every coin in the system has its associated condition, including the default constant event outcome (always true or always false). All the transactions apply the same protocol for proof generation and verification. There is no difference between transactions with event conditions and transactions without event conditions.

In addition, users should not be able to learn any information that can connect conditional coin transactions with event definitions or event outcomes. Both are used as witness information by the provers (private data only accessible to the provers) to generate zero-knowledge proofs. The zero-knowledge protocol verifies existence of the claimed event definition and event outcome against the two Merkle tree roots. It achieves this goal by verifying a Merkle tree path from the leaf record to the root for both the event definition and event outcome embedded in a conditional coin UTXO note. If we assume that security properties hold for the underlying zero-knowledge proof system, then confidentiality of the association between a conditional coin and the corresponding event outcome is guaranteed.

In case of balance, in conditional coin transactions, a coin can be split into two coins of equal value (equals with the summed input coin value), conditional to the opposite outcome of the same event. According to the protocol design, this transaction is allowed only when the input coins are shown to satisfy event outcomes (one of the zero-knowledge proof constraints). Because the event outcome is binary and the decentralized ledger only accepts one event outcome per event definition, balance is also guaranteed. In a conditional coin transaction that the total input coin value is not divided, using zero-knowledge proof, the prover must show that the two output coin notes of the same value satisfy the constraint that they embed the opposite outcomes of the same event as conditions. Using proof by contradiction, if the balance requirement is violated, this means that either the underlying zero-knowledge proving system is broken, or the conditional coin transactions are malleable, or the constraint that one outcome per event is not satisfied. Any one of these contradicts against the assumptions that the underlying zero-knowledge proof system (zk-SNARK in this work) is secure; the system abides by the constraints - one outcome per event; and non-malleability of coin transactions that are already proven in the case of Zcash transaction protocol design [4]. Note that due to the asynchronous nature of conditional coin transactions, the system allows circulation of worthless coins.

7 Performance Optimization

In the default setting of the shielded conditional coin transactions described earlier, records are verified against the entire Merkle hash trees. This achieves maximum privacy with a cost of performance. Verification of Merkle hash tree path from a leaf node to the root is extremely expensive using zero-knowledge proof. In case of Zcash, it supports both shielded and non-shielded transactions. For non-shielded transactions, no additional privacy protection is provided. Coins are transferred in a way very much the same as Bitcoin transactions.

7.1 Merkle Subtree

According to observation of crypto-currency transactions in the real-world, spent coins in a block are most likely recently created (e.g., [29]). This leads to a performance optimization idea that instead of verifying against the entire Merkle hash tree of a ledger, one can verify against a Merkle subtree that covers only the recent transaction records (a time in history until now).

In a simple design with Merkle subtree verification, nodes in the network maintain two root values (double root setting), one for the entire Merkle hash tree and the second one for a subtree of the recent records. To hide a transaction among the recent history of records in a ledger, one can generate a zero-knowledge proof against the Merkle subtree, which could significantly improve performance. In case that a transaction accesses records outside the

[2]The system can apply reputation rating for event publishers, or adopt negative incentives to ensure desirable behaviors such as require event publishers to pay a deposit.

[3]In our system, a repetitive event will create a new inherited event definition each time after the outcome is declared.

[4]Our design extends privacy coin note format and applies the same signature scheme in Zcash to prevent transaction malleability.

history covered by a Merkle subtree, the system can switch back to the full Merkle tree verification mechanism. This design achieves optimization by improving efficiency and performance for the most common spending cases. The level of privacy protection is acceptable considering the number of leaf records that can be covered by a balanced N-ary tree with the height value such as 24 or 32. In case of binary tree, the number of leaves would be 16M or 4G. As a reference, Bitcoin has around 330,000 daily transactions in December 2020.

7.2 K-anonymity

A further optimization is to apply K-anonymity [38], where an accessed record can be hidden among $k-1$ randomly picked records in a ledger. In case of UTXOs, one can hide the UTXO that is to be spent among k randomly selected valid UTXOs recorded by a distributed ledger. The size of k can be pre-defined as a constant. Several deployed crypto-currency systems apply k-anonymity, for instance, Monero [31] and Zether [9]. At high level, crypto-currency mixers are also based on the principle of K-anonymity.

The protocols in Section 5 need slight modification to support K-anonymity. When generating a proof, public input data needs to be extended to include extra K conditional coin commits. These can be randomly selected by a user. A secret boolean vector will be used to record which input coin commits are the ones that the user wants to spend. The proving circuit will take this secret boolean vector as a witness to generate transaction proof. The proving circuit will prove that among the set of input coins, the vector is applied to select the coins that are to be spent in a transaction.

To sum up, a performance optimized design could apply K-anonymity to conditional coin commits, and Merkle subtree verification to the event ledgers. This way, it achieves acceptable privacy protection (based on K-anonymity) with reduced zero-knowledge proof overhead.

8 Implementation and Evaluation

Zero-knowledge proving system allows anyone to verify a transaction without complete information from a prover. In common, zero-knowledge proving system requires a circuit to generate proofs and verifying keys for the verifiers. There are tools and libraries available to achieve this goal (e.g., zk-SNARK [39], Geppetto [12], xJsnark [24]).

8.1 Implementation

In the experiment implementation, we used xJsnark [24]. xJsnark is a tool that achieves "program-to circuit" conversion, i.e., to compile a user-supplied program described in a Java-like source language into a compact circuit representation that is recognized by the existing SNARK libraries. It creates circuits [34] in a libsnark compatible format so that the resulting SNARK can be executed using the libsnark back end [25]. xJsnark implements several optimization to improve efficiency of frequent operations and reduce circuit size. It supports efficient short and long integer arithmetic and implements global optimizations for integer arithmetic. The compiler has a built-in optimizer of circuit minimization. xJsnark has developed a Zcash transaction implementation, which is used in this study as a baseline. As reported in the xJsnark paper, the Zcash circuit implemented in xJsnark is very efficient. The compiled circuit under xJsnark is slightly better than the manually optimized implementation. The reason is due to the low-level arithmetic optimizations and the circuit minimizer.

Table 3: Inputs and witness in addition to the baseline implementation (In: input; Wits: Witness)

	Default setting
In	root of the event definition Merkle tree, root of the event outcome Merkle tree, and block height
Wits	event definitions (input coins, output coins), event outcomes (input coins)
	Double root setting
In	Merkle tree root of the recent N coin commits, roots of the event definition Merkle tree (one root for the entire set and one for the last N definitions), roots of the event outcome Merkle tree (one root for the entire set and one for the last N outcomes), and block height
Wits	event definitions (input coins, output coins), event outcomes (input coins)
	K-anonymity setting
In	a set of random K coin commits, roots of the event definition Merkle trees (one root for the entire set and one for the last N definitions), roots of the event outcome Merkle tree (one root for the entire set and one for the last N outcomes), and block height
Wits	A boolean vector that specifies coins in the set of K that are spent, event definitions (input coins, output coins), event outcomes (input coins)

We extended the Zcash implementation in xJsnark according to the described design to support events and conditions, which includes K-anonymity options. A proving system is used to generate a proof for each JoinSplit transaction based on the extended data models with event conditions and algorithms. xJsnark builds on top of libsnark. Note that in libsnark, we need to generate a key pair of $< pk, vk >$ in a ZK proving system. Eventually, a verifier just needs the vk, public inputs χ, and the proof π to verify a conditional coin transaction.

The system implements three Merkle trees, for the conditional coins, the event definitions, and the event outcomes. In addition to public input, output, and witness defined in Zcash, we made extensions for different settings as listed in Table 3.

All the key parameters like Merkle tree height, time stamp, event ID, event outcome ID, etc are 64-bit size. In the beginning, the circuit verifies the validity of different transaction scenarios by checking both the input event conditions and the output event definitions. For instance, a conditional coin can be transferred with the event condition kept intact. In case a new event condition is attached, the circuit verifies the event outcome announcements associated with the input coins and the event definition for the output coins. In addition, the proving circuit verifies the conditions that are common for all the transaction scenarios.

Integration with consensus protocol: When implementing a UTXO based transaction system, it is common to separate the underlying consensus protocol (e.g., Proof-of-Work, Proof-of-Stake, Practical BFT) from the component that creates and verifies transactions, which is the focus of this work. These two parts are orthogonal to each other. In principle, the implemented system for supporting transaction creation and verification can be integrated with any consensus protocol. For instance, in Zcash, the clean separation of the transaction model and the underlying PoW mechanism allows the system to switch to a different consensus protocol. The decision to pick a consensus protocol often depends on factors that are outside the interest of this work, for instance, permissionless, incentive mechanism, crypto-economic considerations, blockchain parameter optimization, scalability. Therefore, we don't mandate or restrict our implementation to a specific consensus protocol.

Our core system for creating and verifying the shielded conditional UTXO transactions can be integrated with any consensus mechanism that designers opt to choose, in such a way that is similar to how Zcash integrates its transaction protocol implementation with Proof-of-Work consensus.

8.2 Evaluation

For the reasons above, the focus of evaluation is on performance of the core system that generates and verifies transactions. The experiments were conducted using an Intel Xeon computer that has 8 Xeon cores and 64GB ram and running on Ubuntu 18.04. The two key components are the circuit for generating transaction proofs (run by the nodes that propose transactions) and the circuit for verifying the proofs (run by the nodes that validate transactions) described in Section 5. We applied similar performance metrics used in the original Zcash paper for evaluating the extended transaction protocols. These include circuit sizes, key lengths, QAP (Quadratic Arithmetic Programs) degrees, memory overheads for generating proofs and verifying proofs, execution time, etc. The degree of QAP is a metric to measure the complexity of a circuit [8]. Results are collected and reported in this subsection using these well accepted metrics for evaluating a design using zero-knowledge proving system. Note that since our transaction protocols are agnostic to the underlying layers for achieving consensus (discussed at the end of Section 8.1), evaluation does not focus on data like end-to-end latency, overall transaction throughput. These metrics often depend on the design characteristics of the underlying consensus mechanism as well as blockchain parameters such as block delay and block size. The implemented event based transaction system can be applied as a drop-in replacement in Zcash, which cleanly separates transaction processing from consensus processing. Regarding the overall performance, when integrating with the same PoW mechanism and choosing identical parameters as the Zcash implementation, the system achieves comparable performance such as block rate and throughput. It is well known that in zkSNARK, generating proofs is resource intensive and time consuming while verifying proofs is very fast and takes negligible time. This is also reflected in our results below.

Default setting: In the default setting, a single Merkle hash tree is applied respectively for the ledger of the conditional coin commitments, the event definitions, and the event outcomes. For each ledger, the corresponding Merkle tree covers the entire ledger with 64 levels as tree height. To determine complexity, size of the produced circuit was measured, as well as the proving key and the verifying key. The total number of constraints is 11781616 (without optimization) and 8792171 (with xJsnark optimization), respectively. The degree of corresponding QAP is 8912896. The size of the proof key is 16767402202 bits, and the size of the verification key is 29468 bits. Comparing with the implementation of privacy coins without event support [5], the expansion of the circuit size is primarily due to verification of additional Merkle tree hash paths (event definitions and event announcements). The circuit size is overwhelmingly dominated by the Merkle hash tree path verification (more than 95% of the proving circuit is used for verification of the Merkle tree hash paths). The proof size in bits is 2294. It takes on average 261 seconds to compute proof, and verification takes on average 0.042 second. Due to circuit size increase, the proof generation time almost doubles on average. This is because in the Zcoin protocol, verification of Merkle tree hash paths is only applied to the coin commitments whereas in the event based transaction system, it is additionally applied to the event definitions and the even announcements. As the results indicate, average verification time is almost the same with versus without event privacy support. To reduce the overhead of proof generation, we have developed further optimizations based on K-anonymity and Merkle subtree verification, the results are reported below.

[5]The original design implements the default Zcoin protocol. According to the xJsnark developers, this implementation outperforms the manually developed Zcoin implementation by the Zcash team due to various circuit optimizations by the xJsnark toolchain.

Table 4: Circuit implementation results (subtree setting)

	subtree height (32)	subtree height (48)
Constraints	6291056	9036336
Constraints (after xJsnark optimization)	4706490	6749337
QAP degree	4718592	8388608
PK size in bits	8960790003	13265117446
VK size in bits	29468	29468
Peak memory usage (MB)	14932	21186

Double root setting: In this setting, for each ledger, the system maintains two roots, one for the entire Merkle tree (height 64), and the other one for a subtree that covers the most recent records (a height value less than 64). After a transaction is verified, both roots will be updated. The support of subtree can significantly reduce circuit complexity and improve performance as mentioned earlier that performance is predominated by the zero-knowledge Merkle tree verification. For a transaction, if the input records fall within the subtrees, user can opt to generate a proof against the Merkle subtrees instead of the entire Merkle trees. In this work, we assume one subtree for each ledger (coin commitments, event definitions, event outcomes). It is straightforward to extend the concept to multiple subtrees.

In case of proofs against the subtrees, user's transactions are hidden among these that are covered by the subtrees. In practice, the protection is acceptable due to the large size of the subtrees, for instance, 24 or 32 as tree height. It is plausible to reduce the subtree height to a value less than 24, which will further improve performance by a significant amount – which reflects a trade-off between level of privacy and performance. In this work, we only evaluated subtree settings of 32 and 48 as the subtree heights, and the results are shown in Table4.

K-anonymity setting: This is the setting that supports k-anonymity. In case of K-anonymity, user's input coin is hidden among k coins. K-anonymity design provides significant performance advantage meanwhile offers acceptable privacy in practice. In our implementation, the system supports a set with constant number of input coin commitments (k is the size). A secret boolean vector is used to indicate which coins in the set are the ones that a user wants to spend in a transaction. The boolean vector is used as witness when generating a transaction proof. Our experiment shows that proving circuit size is not much affected by the size of K. For instance, with everything else kept the same, changing k from 16 to 32 only adds 249 constraints after optimization (over 2.4 million constraints). For this reason, in the rest of experiment, we fixed k to 64.

Table 5: K-anonymity circuit implementation results

	subtree height (24)	subtree height (32)	subtree height (48)	tree height (64)
Constraints	3283986	4111794	5767410	7423026
Constraints (after xJsnark optimization)	2471632	3089298	4322748	5557005
QAP degree	2621440	3145728	4325376	6291456
PK size in bits	4747036 863	5901516 593	8240801 363	1078180 7920
VK size in bits	111132	111132	111132	111132
Peak memory usage (MB)	7834	9851	13776	17528

According to the description in Section 7, in this setting, K-anonymity is applied to the zero-knowledge verification of coin commits, and Merkle subtrees are used for the event verification. In our experiment, we evaluated four Merkle subtree settings for events (subtree height 24, 32, 48, and complete tree height 64), and the results are shown in Table 5. In case of 64 as tree height, event records are verified against the entire event ledgers.

As the results indicate, under the K-anonymity setting and Merkle subtree (height 24 or 32), it takes on average about 95 seconds (height 24) or 110 seconds (height 32) to generate proof and 0.006 second (both cases) to verify the proof. After these optimizations, proof generation time is on a par with the Zcash implementation. Note that Zcash supports faster non-shielded transactions by turning off privacy protection. Users who do not care privacy can choose non-shielded transactions. In our implementation, all the transactions are shielded transactions with privacy protection. K-anonymity setting and Merkle subtree offer more flexible and finer grained control to the users regarding the trade-off between privacy protection and performance than Zcash. For the non-shielded transactions, both systems would be the very similar with the only difference that the event based UTXO notes and transactions are tagged with event definitions and outcomes.

9 Related Work

Main related work can be categorized into: (i) *Conditional coin research before blockchains*: The concept of electronic cash with its value conditional to event outcome can be found in the literature (e.g., [40]) before Bitcoin was introduced [30]. These schemes do not rely on distributed ledgers in their designs. (ii) *Smart contract with event triggers*: Cryptocurrencies can be deposited to a smart contract. Later, depending on the result of an event based trigger, the locked crypto-assets can be re-distributed. The work with "mixicles" [20] shows us that smart contracts whose results are dependent on the occurrence of the real-world event must employ Oracles as "triggers" for distribution of the funds. One main distinguishing property of confidential conditional coins is that coins with event conditions attached can be treated as regular privacy based UTXOs, thus can be transferred, split, and traded (support for transferability and non-trigger based transaction processing asynchronous to the event handling). There is no trigger in the system and no lockup of assets unlike the previous work [20] where tumblers are triggered by the Oracle inputs. Another key distinguishing property is that we provide full privacy protection of the events embedded in the event based UTXOs with zero-knowledge proof. (iii) *Oracles*: Oracles [22] relay authenticated data from the external sources to blockchain-based systems so that the data can be used by smart contracts, for event triggers. Oracle can leverage trusted third parties, MPC (e.g. [43], [26]), TEEs (e.g., [42], [37]), etc. In the context of conditional coins, Oracle services relate to the event processing and validation of the external data. These efforts are complementary or orthogonal to the main privacy preserving conditional coin protocol design in this work. (iv) *Contingent payments*: In [10], the authors showed how to perform zero-knowledge contingent payments (ZKCP) in Bitcoin securely. The focus of this work and ZKCP is different as the main objective of our protocols is to enable privacy preserving and transferable event based UTXOs with their values conditional with respect to the event outcomes. It applies the approach that decouples event processing from shielded UTXO transaction processing, distinguishing itself from ZKCP. It is worth mentioning that our privacy preserving event based UTXO protocols can support the use cases of contingent payments. (v) *Privacy coins*: The related work includes implementations of confidential cryptocurrencies and assets ([9, 27, 32]). The conditional coins are intended as an extension or to be integrated with the confidential UTXO based assets to enable use cases where values of the confidential assets are conditional to the event outcomes.

10 Challenges and Future Work

The currently demonstrated system has certain limitations. The event conditions are in a binary format. Although complex events can be encoded using boolean logic on top of binary event outcomes, how to support flexible and efficient expression of complex events without significant increase of the proof circuit complexity remains a research topic for future extension of the described system. Moreover, the current implementation based on xJsnark inherits all the limitations of libsnark which is based on the zk-SNARK proving scheme used by Zcash, such as secure setup. Since our protocols described in Section 5 are proving system agnostic, it is straightforward to adapt the transaction protocols to other zero-knowledge proving system to eliminate these limitations, like bulletproof [7]. In addition, further optimization of the proving system could be realized, for instance using a dedicated zero-knowledge proof scheme. To summarize, future work includes (1) support of rich and complex event processing such as event reasoning, (2) proposition logic for the conditional coins, (3) more efficient implementation of the proving circuit and experimenting with other zero-knowledge proving systems (e.g., bulletproof).

11 Conclusion

This paper develops a confidential blockchain and event based transaction system with privacy protection, which utilizes zero-knowledge proof technology. Extending the Zcash data model, the conditional event based transaction protocol design encodes event outcome as part of the privacy (shielded) UTXO notes. A main advantage of such design is that, without relying on any triggering mechanism, the system separates event processing from the transaction processing of the conditional privacy UTXOs. The event based UTXOs can be transferred freely with their values conditional to the linked event outcomes. Event privacy is protected with a shielded transaction protocol design using zero-knowledge proof. The designed protocols are implemented and tested using xJsnark.

References

[1] H. Al-Breiki, M. H. U. Rehman, K. Salah, and D. Svetinovic. 2020. Trustworthy Blockchain Oracles: Review, Comparison, and Open Research Challenges. *IEEE Access* 8 (2020), 85675–85685. https://doi.org/10.1109/ACCESS.2020.2992698

[2] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. 2018. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046. https://eprint.iacr.org/2018/046.

[3] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. 2014. Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture. In *23rd USENIX Security Symposium (USENIX Security 14)*. USENIX Association, San Diego, CA, 781–796. https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/ben-sasson

[4] Manuel Blum, Paul Feldman, and Silvio Micali. 1988. Non-interactive zero-knowledge and its applications. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*. ACM, 103–112.

[5] Sean Bowe, Ariel Gabizon, and Matthew D. Green. 2017. A multi-party protocol for constructing the public parameters of the Pinocchio zk-SNARK. Cryptology ePrint Archive, Report 2017/602. https://eprint.iacr.org/2017/602.

[6] Sean Bowe, Ariel Gabizon, and Ian Miers. 2017. Scalable Multi-party Computation for zk-SNARK Parameters in the Random Beacon Model. Cryptology ePrint Archive, Report 2017/1050. https://eprint.iacr.org/2017/1050.

[7] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. 2018. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy*. 315–334.

[8] Vitalik Buterin. 2016. Quadratic Arithmetic Programs: from Zero to Hero.

[9] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. 2019. Zether: Towards Privacy in a Smart Contract World. Cryptology ePrint Archive, Report 2019/191. https://eprint.iacr.org/2019/191.

[10] Matteo Campanelli, Rosario Gennaro, Steven Goldfeder, and Luca Nizzardo. 2017. Zero-knowledge contingent payments revisited: Attacks and payments for services. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 229–243.

[11] Ting Chen, Zihao Li, Yuxiao Zhu, Jiachi Chen, Xiapu Luo, John Chi-Shing Lui, Xiaodong Lin, and Xiaosong Zhang. 2020. Understanding Ethereum via Graph Analysis. *ACM Trans. Internet Technol.* 20, 2, Article 18 (April 2020), 32 pages. https://doi.org/10.1145/3381036

[12] Craig Costello, Cédric Fournet, Jon Howell, Markulf Kohlweiss, Benjamin Kreuter, Michael Naehrig, Bryan Parno, and Samee Zahur. 2015. Geppetto: Versatile verifiable computation. In *2015 IEEE Symposium on Security and Privacy*. IEEE, 253–270.

[13] Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. 2001. Robust non-interactive zero knowledge. In *Annual International Cryptology Conference*. Springer, 566–598.

[14] Christina Garman, Matthew Green, and Ian Miers. 2016. Accountable privacy for decentralized anonymous payments. In *International Conference on Financial Cryptography and Data Security*. Springer, 81–98.

[15] Oded Goldreich and Ariel Kahan. 1996. How to construct constant-round zero-knowledge proof systems for NP. *Journal of Cryptology* 9, 3 (1996), 167–189.

[16] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. 1985. The knowledge complexity of interactive proof-systems. In *Proceedings of the 7th annual ACM Symposium on Theory of Computing - STOC 1985*. ACM, 291–304.

[17] Jens Groth, Rafail Ostrovsky, and Amit Sahai. 2006. Perfect non-interactive zero knowledge for NP. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 339–358.

[18] Suyash Gupta, Jelle Hellings, Sajjad Rahnama, and Mohammad Sadoghi. 2020. Blockchain Consensus Unraveled: Virtues and Limitations. In *Proceedings of the 14th ACM International Conference on Distributed and Event-Based Systems* (Montreal, Quebec, Canada) *(DEBS '20)*. Association for Computing Machinery, New York, NY, USA, 218–221. https://doi.org/10.1145/3401025.3404099

[19] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. 2016. Zcash protocol specification. *Tech. rep. 2016–1.10. Zerocoin Electric Coin Company, Tech. Rep.* (2016).

[20] Ari Juels, Lorenz Breidenbach, Alex Coventry, Sergey Nazarov, Steve Ellis, and Brendan Magauran. 2019. Mixicles: Simple Private Decentralized Finance.

[21] Kamil Kluczniak and Man Ho Au. 2018. Fine-Tuning Decentralized Anonymous Payment Systems based on Arguments for Arithmetic Circuit Satisfiability. *IACR Cryptol. ePrint Arch.* 2018 (2018), 176.

[22] Petar Kochovski, Sandi Gec, Vlado Stankovski, Marko Bajec, and Pavel D Drobintsev. 2019. Trust management in a blockchain based fog computing platform with trustless smart oracles. *Future Generation Computer Systems* 101 (2019), 747–759.

[23] A Kosba, A Miller, E Shi, Z Wen, et al. 2015. Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts, Tech. In *2016 IEEE Symposium on Security and Privacy (SP). Available at: https://ieeexplore. ieee. org/-document/7546538*.

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

[25] libsnark 2012-2017. libsnark: a C++ library for zkSNARK proofs. https://github.com/scipr-lab/libsnark.

[26] José María Manzano, JM Nadales, D Muñoz de la Peña, and Daniel Limón. 2019. Oracle-Based Economic Predictive Control. In *2019 IEEE 58th Conference on Decision and Control (CDC)*. IEEE, 4246–4251.

[27] Greg Maxwell. 2016. Confidential transactions. https://people.xiph.org/~greg/.

[28] Ian Miers, Christina Garman, Matthew Green, and Aviel D Rubin. 2013. Zerocoin: Anonymous distributed e-cash from bitcoin. In *2013 IEEE Symposium on Security and Privacy*. IEEE, 397–411.

[29] Malte Möser, Kyle Soska, Ethan Heilman, Kevin Lee, Henry Heffan, Shashvat Srivastava, Kyle Hogan, Jason Hennessey, Andrew Miller, Arvind Narayanan, and Nicolas Christin. 2018. An Empirical Analysis of Traceability in the Monero Blockchain. *Proceedings on Privacy Enhancing Technologies* 2018 (06 2018), 143–163. https://doi.org/10.1515/popets-2018-0025

[30] Satoshi Nakamoto. 2019. *Bitcoin: A peer-to-peer electronic cash system.* Technical Report. Manubot.

[31] Shen Noether. 2015. Ring SIgnature Confidential Transactions for Monero. *IACR Cryptology ePrint Archive* 2015 (2015), 1098.

[32] Shen Noether and Adam Mackenzie. 2016. Ring Confidential Transactions. *Ledger* 1 (12 2016), 1–18. https://doi.org/10.5195/LEDGER.2016.34

[33] B. Parno, J. Howell, C. Gentry, and M. Raykova. 2013. Pinocchio: Nearly Practical Verifiable Computation. In *2013 IEEE Symposium on Security and Privacy*. 238–252. https://doi.org/10.1109/SP.2013.47

[34] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. 2016. Pinocchio: Nearly Practical Verifiable Computation. *Commun. ACM* 59, 2 (Jan. 2016), 103–112. https://doi.org/10.1145/2856449

[35] F. Reid and M. Harrigan. 2011. An Analysis of Anonymity in the Bitcoin System. In *2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing*. 1318–1326. https://doi.org/10.1109/PASSAT/SocialCom.2011.79

[36] Dorit Ron and Adi Shamir. [n.d.]. Quantitative Analysis of the Full Bitcoin Transaction Graph.

[37] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. 2015. Trusted execution environment: what it is, and what it is not. In *2015 IEEE Trustcom/BigDataSE/ISPA*, Vol. 1. IEEE, 57–64.

[38] Pierangela Samarati and Latanya Sweeney. 1998. *Protecting Privacy when Disclosing Information: k-Anonymity and its Enforcement through Generalization and Suppression.* Technical Report. SRI International.

[39] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. 2014. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*. IEEE, 459–474.

[40] Larry Shi, Bogdan Carbunar, and Radu Sion. 2007. Conditional E-Cash. In *Financial Cryptography and Data Security*, Sven Dietrich and Rachna Dhamija (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 15–28.

[41] Yonatan Sompolinsky, Yoad Lewenberg, and Aviv Zohar. 2016. SPECTRE: A Fast and Scalable Cryptocurrency Protocol. *IACR Cryptol. ePrint Arch.* 2016 (2016), 1159.

[42] Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. 2016. Town Crier: An Authenticated Data Feed for Smart Contracts. Cryptology ePrint Archive, Report 2016/168. https://eprint.iacr.org/2016/168.

[43] Fan Zhang, Sai Krishna Deepak Maram, Harjasleen Malvai, Steven Goldfeder, and Ari Juels. 2019. DECO: Liberating Web Data Using Decentralized Oracles for TLS. *arXiv preprint arXiv:1909.00938* (2019).