University of Texas Rio Grande Valley

ScholarWorks @ UTRGV

Computer Science Faculty Publications and Presentations

College of Engineering and Computer Science

6-2021

On Conditional Cryptocurrency With Privacy

Rabimba Karanjai

Lei Xu
The University of Texas Rio Grande Valley, lei.xu@utrgv.edu

Zhimin Gao

Lin Chen

Mudabbir Kaleem

See next page for additional authors

Follow this and additional works at: https://scholarworks.utrgv.edu/cs_fac

Recommended Citation

R. Karanjai, L. Xu, Z. Gao, L. Chen, M. Kaleem and W. Shi, "On Conditional Cryptocurrency With Privacy," 2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), 2021, pp. 1-3, doi: 10.1109/ICBC51069.2021.9461133.

This Conference Proceeding is brought to you for free and open access by the College of Engineering and Computer Science at ScholarWorks @ UTRGV. It has been accepted for inclusion in Computer Science Faculty Publications and Presentations by an authorized administrator of ScholarWorks @ UTRGV. For more information, please contact justin.white@utrgv.edu, william.flores01@utrgv.edu.

Authors
Rabimba Karanjai, Lei Xu, Zhimin Gao, Lin Chen, Mudabbir Kaleem, and Weidong Shi

This conference proceeding is available at ScholarWorks @ UTRGV: https://scholarworks.utrgv.edu/cs_fac/69

On Conditional Cryptocurrency With Privacy

ABSTRACT

In this paper, we present the design and implementation of a conditional cryptocurrency system with privacy protection. Unlike the existing approaches that often depend on smart contracts where cryptocurrencies are first locked in a vault, and then released according to event triggers, the conditional cryptocurrency system encodes event outcome as part of a cryptocurrency note in a UTXO based system. Without relying on any triggering mechanism, the proposed system separates event processing from conditional coin transaction processing where conditional cryptocurrency notes can be transferred freely 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 conditional assets and prevents assets from being locked. In this work, we demonstrate a method of confidential conditional coin by extending the Zerocoin data model and protocol. The system is implemented and evaluated using xJsnark.

1 INTRODUCTION

The primary objective of this work is to enable cryptocurrency notes (e.g., UTXO based coins) with conditional values, specifically for pri-vacy coins (e.g., Zcash [22]). Although connecting cryptocurrency transactions with events [24] (such as issuing transactions based on event triggers) is not new, the proposed approach distinguishes from the prior efforts in several aspects. First, it associates event conditions with UTXO based privacy coins, demonstrated using Zcash as a targeted system, which is named as conditional privacy coins. Second, the links between events and conditional coins are confidential to achieve condition privacy. Using a zero-knowledge based protocol, a conditional coin can be validated against the as-sociated event outcome without disclosing which event that it has been linked to. Third, conditional coins can be transferred and traded freely in the system before the associated event has occurred or its outcome has been declared (asynchronous). There is 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 con-ditional coin, not a transaction of the coin, that is conditional to event outcome. This has significant benefits over some existing or alternative designs that freeze assets in a vault or smart contract before event result is published. Fourth, the system decouples event processing and transaction processing. This mechanism facilitates conditional coin processing and allows transactions of conditional coins to be validated in an asynchronous manner from the event handling by the system. 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 propose a conditional privacy cryptocurrency system design where confidential event conditions are associated with privacy coins. (ii) The proposed design allows conditional coins to be transferred, exchanged, and traded freely in an asynchronous manner from event processing logic. There is no asset lock-up or freeze of conditional coins in a vault that a release is triggered in accordance with event results. It is the value of the coins that is conditional to event outcome. (iii) We implement the conditional privacy coin protocol in xJsnark with Zcash as a reference model.

2 OVERVIEW

Problem statement: To enable private and transferable conditional coins, the system needs to meet the following requirements:

- Privacy coins (e.g., use Zcash as a base model and initial target) are associated with event conditions. Conditional coins can be validated against event outcomes. However, links between conditional coins and events (both event definitions and event outcome announcements) are hidden from the validators.
- The system should support complete transferability of 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.
- Event processing and transaction processing are decoupled. There is a separation between event processing and conditional coin transactions. Event processing includes registration of events and event outcome announcements/declarations. Updates to each part of the decoupled system are asynchronous.
- All the coins in the system, regardless if they have event conditions attached or not, are indistinguishable. The system does not treat 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.

It is worth mentioning that certain things are outside the scope of this paper. These include:

- Validation of real-world event outcome which can be done with several approaches described in the literature such as using TEEs, zero-knowledge proof, multi-party computation, etc;

- Oracle service, which is an orthogonal topic (Oracle service can be integrated with event processing and our system can be made to support any type of Oracle service design);
- Design of sophisticated event logic - a subject related to event processing and a possible direction of future research.

In addition, to make things simple, we assume in this paper 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, logisitcs, 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, first, we adopt a system, where event processing is decoupled from the transaction processing. There are separate logical chains of records for registering event definitions, announcements of 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 commitments of the conditional 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 question, which could be supported through a variety of approaches such as Oracle service, a group of event validators (with tolerance of Byzantine failures), multi-party computation, or zero-knowledge proof based claim verification scheme (e.g., [26]). 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. A privacy coin without a real condition is configured to use one of the default constant events as its condition. This way, all the transactions involving conditional coins in the system (payments using conditional coins or transferring of conditional coins) follow the same 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 coins 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 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 coins.

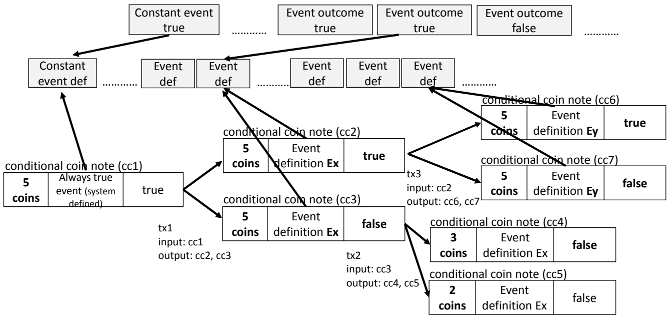

Figure 1: Illustration of conditional coin transactions.

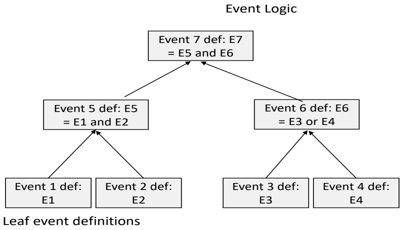

Figure 2: Support for complex event logic and processing.

Conditional coins can be transferred as demonstrated by tx2 in Figure 1. To remove the condition, the coin needs to be spent with a matched event outcome, see tx3. Validation is done using zero-knowledge proof without leaking the associated event definition and event outcome. After the condition is removed, a conditional coin 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 as shown in Fig 2. In addition, event processing can be augmented with smart contract support to enable more diverse means for validating event outcomes.

Applications: Privacy preserving conditional coins may enable and facilitate a wide range of applications including but not limited to, transferable confidential assets with valuation based on event outcomes, bill of exchange based on crypto-assets, future contracts, prediction market, and use case areas such as logistics, FinTech, insurance (such as enable secondary market), etc.

3 DATA MODELS

The data models are based on extending the notations of Zcash [8, 17]. In this paper, for simplification, we apply zk-SNARK and the original Zcash design [8, 17] 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 [1], bulletproof [4]. Discussion leveraging different back-end zero-knowledge proving systems is orthogonal to the scope of this work, which

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

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
	Conditional coin 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)

focuses primarily on the front-end protocol design of a blockchain based privacy preserving transaction system for conditional coins.

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 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 that adopts Proof-of-Work for blockchain consensus. Although our data models and protocol are based on extending Zcash specification, the design can be adapted to any blockchain based systems, for instance substituting PoW consensus with Proof-of-Stake (PoS) or different flavors of BFTs.

There are three main types of data records: (i) records for event registration and definition; (ii) records for event outcome declaration; and (iii) records for conditional coin payment and transferring. For simplicity, we assume in this work that all the records are managed by a single ledger. However, since event processing is decoupled from the conditional coin transactions, it is plausible to track these records separately, using sub-chains or sub-ledgers. For instance, a system can employ a sub-chain for event registration and event declaration/announcement, and another sub-chain for conditional coin transactions. A global chain could be used to coordinate these sub-chains in way similar to the concept of sharding. A key point is that our system does not place restriction on alternative implementations that optimize for throughput and/or parallel transaction processing. Event processing and conditional coin transactions are asynchronous.

The data models are described in Table 1. 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 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. The system can be extended to more complex event outcome scenarios, which is a subject of future work. For the default constant events, their event outcomes are defined in the genesis block as ea^{dft}_{false} and ea^{dft}_{true}.

Payment address and conditional 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 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.

A conditional 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 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.

A conditional note can be spent or transferred without waiting for announcement of the anticipated event outcome. This characteristic, transferability of 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.

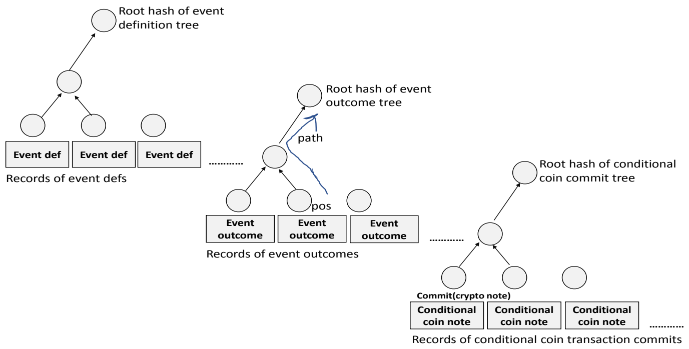
Root hash of event definition tree
Event def
Event def
Event def
Records of event defs
Root hash of event outcome tree
path
pos
Event outcome
Event outcome
Event outcome
Records of event outcomes
Root hash of conditional coin commit tree
Commit(crypto note)
Conditional coin note
Conditional coin note
Conditional coin note
Records of conditional coin transaction commits

Figure 3: Multiple Merkle hash trees for conditional coins, event definitions, and event outcomes.

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. 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 node to the root.

Public parameters: In the case of the experiments in this paper, the system uses the same set of public parameters *pp* in Zcash design. They are generated either by a trusted party at the beginning or through a Multi-Party Computation ceremony [2, 3]. 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 conditional coin payment and transferring to cases of two input notes and two output notes. Payment involving more than two input notes or two output notes can be reduced to a series of transactions, each only 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.

4 PROTOCOL DESIGN FOR CONDITIONAL PRIVACY COINS WITH TRANSFERABILITY

4.1 Preliminary

The protocol extends the design of zero-knowledge based privacy payment system with Zcash as the baseline. Privacy oriented blockchains apply zero-knowledge proving system as the underlying building block to protect privacy. A zero-knowledge proving system is a cryptography protocol that allows proving a particular claim/statement, dependent on two input datasets, public and witness, without disclosing information about the witness input other than that included in the claim/statement.

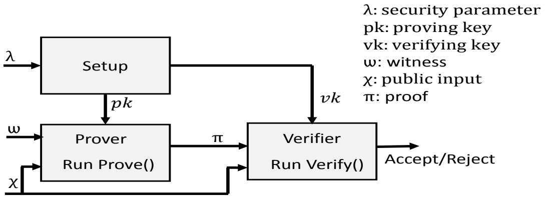

Figure 4: Diagram of zero-knowledge proving system.

A zero-knowledge proving system needs to satisfy the following security requirements: completeness, succinctness, and proof-of-knowledge.

Completeness: for security parameter λ, a $\mathbb{F}$, arithmetic circuit C [20], and any $(\chi, \omega) \in \mathcal{R}$ (satisfying inputs), honest prover can convince the verifier with probability $1 - negl(\lambda)$.

Succinctness: honestly generated proof π has $O_n(1)$ bits and Verify(vk, χ, π) runs in the $O_\lambda(|\chi|)$.

Proof-of-knowledge: If the verifier accepts a proof output by a bounded prover, then the prover knows witness for the given statement.

4.2 Transaction Algorithms

For completeness, this subsection lists the main algorithms. The main extensions are related to the JoinSplit transaction. JoinSplit is extended to support associating a privacy UTXO note with an event outcome.

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}. $pk_{JoinSplit}$ and $vk_{JoinSplit}$ are a pair of proving and verifying keys for JoinSplit transactions. pp_{enc} is encryption key and pp_{sig} is signature key.

Algorithm 1: Public parameter generation algorithm.

```
Input  : Security parameter $\lambda$, proving circuit $C_{JoinSplit}$
Output : Public parameters $pp_{CPC}$
Compute $C_{JoinSplit}$ at security parameter $\lambda$
Compute $(pk_{JoinSplit}, vk_{JoinSplit}) = KeyGen(1^\lambda, C_{JoinSplit})$
Create $pp_{enc} = \mathcal{G}_{enc}(1^\lambda)$
Create $pp_{sig} = \mathcal{G}_{sig}(1^\lambda)$
Output $pp_{CPC} = (pk_{JoinSplit}, vk_{JoinSplit}, pp_{enc}, pp_{sig})$
```

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. The value of *KA.Base*, computation of *KA.FormatPrivate* and *KA.DerivePublic* follow the Zcash implementation.

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 $\perp$

Algorithm 2: Address generation.

```
Output: key pairs: $(a_{pk}, a_{sk})$ and $(pk_{enc}, sk_{enc})$
Set $a_{sk} = \mathsf{PRF}_{a_{sk}}(randomvalue)$
Set $a_{pk} = \mathsf{PRF}^{addr}_{a_{sk}}(0)$
Set $sk_{enc} = KA.FormatPrivate(PRF^{addr}_{a_{sk}}(1))$
Set $pk_{enc} = KA.DerivePublic(sk_{enc}, KA.base)$
Set transmission key as $(a_{pk}, pk_{enc})$
Set receiving key as $(a_{pk}, sk_{enc})$
```

(reject). This algorithm is used to mint notes based on the public value and put the notes on L_{CPC}.

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 3 lists pseudo-code adapted from the Zcash specification. With such transactions, conditional coin notes can be created by spending existing notes and transferred regardless whether the event outcome has been declared.

Algorithm 3: JoinSplit algorithm for conditional privacy coins.

```
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 $a^{old}_{sk,1}$ and $a^{old}_{sk,2}$, 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 $a^{new}_{pk,1}$ and $a^{new}_{pk,2}$, public value $v^{old}_{pub}$, block height $block_n$
Output: A transaction $tx_{join\_split}$ with two output conditional coin notes
For each input note, compute nullifier $nf_i^{old} = \mathsf{PRF}^{nf_{old}}_{a^{old}_{sk,i}}(n_i^{old}.\rho) = SHA256(a^{old}_{sk,i} \| n_i^{old}.\rho)$
Create signature key pair $(pk_{sig}, sk_{sig}) = \kappa_{sig}(pp_{sig})$ where $\kappa_{sig}$ generates a key pair for signature signing
Compute $h_{sig} = \mathsf{CRH}(nf_1^{old}, nf_2^{old}, pk_{sig})$ where CRH is a collision resistant hash function
Set $\rho_i^{new} = \mathsf{PRF}^{\rho}_{\varphi}(i, h_{sig}) = SHA256(i \| \varphi \| h_{sig})$
For each new note, sample a random $\gamma_i^{new}$
Set new conditional note and its attributes as $n_i^{new} = (a^{new}_{pk,i}, v_i^{new}, edef^{new}, cond_i^{new}, \rho_i^{new}, \gamma_i^{new})$
Compute new note commitment $cm_i^{new} = NoteCommit(n_i^{new})$ where NoteCommit is based on SHA256
Compute old note spending signature $h_i = \mathsf{PRF}^{pk}_{a^{old}_{sk,i}}(i || h_{sig})$
Encrypt new note as $N_i^{enc,new} = \mathcal{E}_{enc}(pk^{new}_{enc,i}, n_i^{new})$
Set public input $\chi = (rt^{tx}, rt^{edef}, rt^{ea}, nf_1^{old}, nf_2^{old}, cm_1^{new}, cm_2^{new}, v^{old}_{pub}, v^{new}_{pub}, block_n, h_{sig}, h_1, h_2)$
Set witness $\omega = (n_1^{old}, n_2^{old}, a^{old}_{sk,1}, a^{old}_{sk,2}, \varphi, dummy_1^{old}, dummy_2^{old}, pos^{old}_{n,1}, path^{old}_{n,1}, pos^{old}_{n,2}, path^{old}_{n,2}, ea_1^{old}, ea_2^{old}, pos^{old}_{ea,1}, path^{old}_{ea,1}, pos^{old}_{ea,2}, path^{old}_{ea,2}, n_1^{new}, n_2^{new}, edef^{new}, pos^{new}_{edef}, path^{new}_{edef})$
Compute proof $\pi_{JoinSplit} = Prove(pk_{JoinSplit}, \chi, \omega)$
Set transaction message $m = (\chi, \pi_{JoinSplit}, N_1^{enc,new}, N_2^{enc,new})$
Set message signature $\delta = S_{sig}(sk_{sig}, m)$
Set $tx_{join\_split} = (rt^{tx}, nf_1^{old}, nf_2^{old}, cm_1^{new}, cm_2^{new}, v^{new}_{pub}, *)$ where $* = (pk_{sig}, h_1, h_2, \pi_{JoinSplit}, N_1^{enc,new}, N_2^{enc,new}, \delta)$
```

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

When generating $\pi_{JoinSplit}$, the proving circuit verifies a set of constraints for the generated transaction, which is applied to generate a proof to be checked by a verifier.

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 follows the Zcash design. It scans the ledger and outputs received note for each JoinSplit transaction.

Verify$_{CPC}$: It takes a set of inputs as described in Algorithm 5. It appends $tx_{JoinSplit}$, n_1^{new} and n_2^{new} to the ledger, or output $\perp$.

4.3 Event transactions

Algorithm 6 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}.

5 SECURITY ANALYSIS

The conditional privacy coin protocol is based on extending the data models and transaction design of the original Zcash protocol. Since the conditional coin 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 conditional coins [17][12][7][10]. This means that we can focus on the properties unique to the conditional coin protocol.

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 such as eclipse attack, 51% attack, denial-of-service at network level, side-channel exploits such as using time delay in transactions as side-channel information, privacy leakage through network traffic patterns, attack to DNS, hard forks, quantum attack, and etc. In addition, the system will ensure that event registration and event outcome declaration are

Algorithm 4: Prove algorithm.

```
Input  : $pk_{JoinSplit}, \chi, \omega$
Output : $\pi_{JoinSplit}$
Verify common constraints below:
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 $rt^{tx}$
Nullifier integrity: $nf^{old}_i = \mathsf{PRF}^{nf_{old}}_{a^{old}_{sk,i}}(n^{old}_i.\rho)$
Spending key validity: $a^{old}_{pk_i} = \mathsf{PRF}^{addr}_{a^{old}_{sk,i}}(0)$
Uniqueness of $\rho^{new}_i$: $\rho^{new}_i = \mathsf{PRF}^{\rho}_{\varphi}(i, h_{sig})$
Note commit validity: $cm^{new}_{n,i} = NoteCommit(n^{new}_i)$
Event definition validity in new notes: $n^{new}_1.eID = n^{new}_2.eID = edef^{new}.eID$
Merkle path validity for the event definition associated with the new notes: $(pos^{new}_{edef}, path^{new}_{edef})$ valid tree path from $edef^{new}$ to the Merkle tree root $rt^{edef}$
When $(n^{new}_i.eID \neq n^{old}_i.eID) \vee (n^{new}_i.eID = n^{old}_i.eID \wedge n^{new}_i.eID = (edef^{dft}_{true}.eID \vee edef^{dft}_{false}.eID))$:
    Event condition validity: $n^{old}_i.cond = ea^{old}_i.v$
    Merkle path validity for the event announcements associated with the old notes: $(pos^{old}_{ea,i}, path^{old}_{ea,i})$ valid tree path from $ea^{old}_i$ to the Merkle tree root $rt^{ea}$
Sub-circuits to validate constraints for different conditional coin spending scenarios:
Case: $(n^{new}_i.eID = n^{old}_i.eID) \wedge (n^{new}_i.eID \neq (edef^{dft}_{true}.eID \vee edef^{dft}_{false}.eID))$
    Value validity: $v^{old}_1 + v^{old}_2 = v^{new}_1 + v^{new}_2$
    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
Case: $(n^{new}_i.eID = n^{old}_i.eID) \wedge (n^{new}_i.eID = (edef^{dft}_{true}.eID \vee edef^{dft}_{false}.eID))$
    Value validity: $v^{old}_1 + v^{old}_2 + v^{old}_{pub} = v^{new}_1 + v^{new}_2 + v^{new}_{pub}$
Case: $(n^{new}_i.eID \neq n^{old}_i.eID) \wedge (n^{new}_i.eID \neq (edef^{dft}_{true}.eID \vee edef^{dft}_{false}.eID))$
    Value validity: $(v^{new}_1 = v^{new}_2) \wedge (v^{old}_1 + v^{old}_2 + v^{old}_{pub} = v^{new}_1 + v^{new}_{pub})$
    New note event condition validity: $(n^{new}_1.cond = true \wedge n^{new}_2.cond = false) \vee (n^{new}_2.cond = true \wedge n^{new}_1.cond = false)$
Case: $(n^{new}_i.eID \neq n^{old}_i.eID) \wedge (n^{new}_i.eID = (edef^{dft}_{true}.eID \vee edef^{dft}_{false}.eID))$
    Value validity: $v^{old}_1 + v^{old}_2 + v^{old}_{pub} = v^{new}_1 + v^{new}_2 + v^{new}_{pub}$
```

properly secured. Event outcome is validated by the participating nodes of the decentralized ledger using the same consensus mechanism for the conditional coin transactions. 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 same in the Zcash design, 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); and for each registered event, the system accepts only one result [2]. Last but not the least, we consider that the underlying zero-knowledge proving system is secure.

Algorithm 5: 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
Parse $tx_{join\_split}$
Set $b1 \longleftarrow Verify(vk_{JoinSplit}, \chi, \pi_{JoinSplit})$
Set $b2 \longleftarrow v_{sig}(pk_{sig}, m, \delta)$ where m is transaction message and $\delta$ is message signature defined in Algorithm 3
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 $\mathsf{CRH}(nf^{old}_1, nf^{old}_2, pk_{sig})$
```

Algorithm 6: 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 $edef$
Sample a new eaID
Set $ea = (edef.eID, eaID, v, block_n)$
When $edef.repeat$ is true
    Create a new signature key pair $(pk^{new}_{sig}, sk^{new}_{sig}) = \kappa_{sig}(pp_{sig})$ where $\kappa_{sig}$ generates a key pair for signature signing
    Sample a new $eID^{new}$
    Set $edef^{new} = (eID^{new}, edef.repeat, bt^{new}, edef.ba, pk^{new}_{sig})$
Set transaction message $m = (ea, edef^{new})$
Set message signature $\delta = S_{sig}(sk_{sig}, m)$
Set $tx_{ea} = (m, \delta)$
```

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 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.

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

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-knolwedge 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 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 coins 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 proven in the case of Zcash transaction protocol design [3]. Note that due to the asynchronous nature of conditional coin transactions, the system allows circulation of worthless coins.

6 IMPLEMENTATION AND EVALUATION

Zero-knowledge proving system allows anyone to verify a transaction without complete information from a prover. It means that the system does not need to disclose private transaction information (e.g., event condition, event definition, event outcome, amounts, and etc.). The verifier can still validate if the transaction instance satisfies the preset conditions. 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.

Implementation: In the experiment implementation, we used xJsnark [13]. 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 [20] in a libsnark compatible format so that the resulting SNARK can be executed using the libsnark back end [14]. 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: Key implementation parameters

Attribute	Size
All Merkle tree height	64
Time stamp	64 bits
Event ID	64 bit
Event outcome ID	64 bit
Coin value	64 bits

We extended the Zcash implementation in xJsnark according to the described design to support events and conditions. 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 conditional coins, event definitions, and event outcomes. In addition to public input, output, and witness defined in Zcash, we extended with the following input and witness:

- **Additional Input:** root of event definition Merkle tree, root of event outcome Merkle tree, and block height;
- **Additional Witness:** event definitions (input coins, output coins), event outcomes (input coins).

Some key parameters are listed in Table 3. In the beginning, the circuit verifies the validity of different transaction scenarios by checking both input event conditions and 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 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.

Circuit evaluation: The experiments were conducted using an Intel Xeon computer that has 8 Xeon cores and 64GB ram and running on Ubuntu 18.04. Size of the produced circuit was measured, as well as proving key and 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. 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 dominated by Merkle hash tree path verification. Future implementation and evaluation could employ k-anonymity and other zero-knowledge proof system to improve speed and efficiency.

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

The proof size in bits is 2294. It takes on average 261 seconds to compute proof, and verification takes on average 0.042 second.

7 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., [23]) before Bitcoin [18]. 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" [9] 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. The main distinguishing property of conditional privacy coins is that coins with event conditions attached can be treated as regular privacy coins, thus can be transferred, split, and traded (support for transferability and non-trigger based transaction processing asynchronous to event handling). There is no trigger in the system and no lockup of assets unlike the previous work [9] where tumblers are triggered by the Oracle inputs. **(iii)** *Oracles*: Oracles [11] relay authenticated data from external sources to blockchain-based systems so that the data can be used by smart contracts, for instance as event triggers. Oracle can leverage trusted third parties, MPC (e.g. [26], [15]), TEEs (e.g., [25], [21]), etc. In the context of conditional coins, Oracle services mostly relate to event processing and validation of external data. These research efforts are complementary to the main conditional coin protocol design in this work. **(iv)** *Contingent payments*: In [6], the authors showed how to perform zero-knowledge contingent payments (ZKCP) in Bitcoin securely. The focus of conditional coins and ZKCP is different as the main objective of conditional privacy coins is to enable privacy preserving and transferable crypto-assets with their values conditional with respect to event outcomes. It applies the approach that decouples event processing from cryptcurrency transaction processing, which distinguishes from ZKCP. It is worth mentioning that conditional coins can support the use cases of contingent payments. **(v)** *Privacy coins*: Last but not the least, the related work includes implementations of confidential cryptocurrencies and assets (e.g., [5, 16, 19]). As we have demonstrated, conditional coins are intended as an extension or to be integrated with confidential or privacy crypto-currencies to enable use cases where values of confidential coins are conditional to event outcomes, and meanwhile the associations between conditional coins and event registration/event outcomes are kept confidential.

8 CONCLUSION

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

REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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).

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

[10] 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.

[11] 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.

[12] 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.*

[13] 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.

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

[15] 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.

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

[17] 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.

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

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

[20] 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

[21] 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.

[22] 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.

[23] 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.

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

[25] 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.

[26] 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).