Accountable Privacy for Decentralized Anonymous Payments

Christina Garman, Matthew Green, and Ian Miers

Johns Hopkins University
{cgarman, mgreen, imiers}@cs.jhu.edu

Abstract. Decentralized ledger-based currencies such as Bitcoin provide a means to construct payment systems without requiring a trusted bank. Removing this trust assumption comes at the significant cost of transaction privacy. A number of academic works have sought to improve the privacy offered by ledger-based currencies using anonymous electronic cash (e-cash) techniques. Unfortunately, this strong degree of privacy creates new regulatory concerns, since the new private transactions cannot be subject to the same controls used to prevent individuals from conducting illegal transactions such as money laundering. We propose an initial approach to addressing this issue by adding *privacy preserving* policy-enforcement mechanisms that guarantee regulatory compliance, allow selective user tracing, and admit tracing of tainted coins (e.g., ransom payments). To accomplish this new functionality we also provide improved definitions for Zerocash and, of independent interest, an efficient construction for simulation sound zk-SNARKs.

1 Introduction

The success of decentralized currencies like Bitcoin has led to renewed interest in anonymous electronic cash both in academia [2, 9, 20] and in practice (including Coinjoin, CryptoNote, and DarkWallet). It has also highlighted new problems related to trust, privacy and regulatory compliance. In modern electronic payment systems, users must trust that their bank is not tampering with the system (e.g., by "forging" extra currency), that no party is abusing the privacy of users' transactions, and simultaneously, that other users are not using the system to engage in money laundering or extortion. Unfortunately, these goals seem fundamentally at odds.

Decentralized payment systems such as Bitcoin address the first issue by replacing the central bank with a distributed ledger and consensus system. Unfortunately, this benefit comes at a significant cost to privacy [1, 5, 22], since any user can now view the transaction graph and potentially trace payments made by another user. Proposals such as Zerocoin and Zerocash [2, 20] attempt to resolve the privacy problem by employing sophisticated zero knowledge proofs of transaction correctness. While such constructions address privacy concerns, they do not offer any additional protections against money laundering and other activity. From an investigative standpoint, Zerocash is no different than cash.

The problem of preventing money laundering in decentralized currencies is not theoretical. In May 2015 the decentralized payment network Ripple was ordered to pay a $700,000 fine by the U.S. Financial Crimes Enforcement Network (FINCEN) due to inadequate monitoring of transactions on their network. This raises questions for the deployment of privacy-preserving payment networks, where individual network operators and currency exchanges may be held criminally liable for facilitating laundering. In general, the difficulty of preventing abuse may prove a barrier to the deployment of private currency systems. In particular, it seems likely that the application of existing reporting requirements will force exchanges/online wallets, even when dealing with anonymous currencies, to hold vast amounts of information about their users, their transactions, and the amounts. If, as in Bitcoin, most consumers use online wallets, then they gain little privacy.

In this paper we aim for a middle ground. Specifically, we design new Decentralized Anonymous Payment (DAP) systems [2] that are capable of enforcing compliance with specific transaction policies, while protecting the privacy of network participants. Our approach builds on the techniques of Ben-Sasson et al.'s Zerocash system [2], using efficient zero knowledge Arguments of Knowledge (zk-SNARKs) [3, 4, 6, 11, 13, 18, 19, 21] to both prove the validity of transactions and to enforce transaction *policies*. These policies allow network participants to verifiably prove compliance with global transaction policies, such as tax payments, deposit limits, and verifiable coin tracing, all while protecting the privacy of each transaction. Interestingly, a side effect of our mechanisms is that even centralized banks or exchanges participating in our scheme are prevented from evading anti-money laundering controls, as these controls are enforced by cryptographically sound proofs.

Improving security definitions for DAP schemes. A necessary starting point for our exploration is to find a formal security definition for DAP schemes that can be readily extended to incorporate our policies. Previous security definitions for DAP schemes involve a series of distinct games, each of which addresses one aspect of the security requirements for the DAP scheme. Not only are these games complex, they fail to simultaneously ensure both *correctness* and privacy of the DAP scheme. In particular, we note that the Zerocash scheme [2] admits a practical attack due to a limitation of the correctness definition. In this paper we explore a more promising route: we define an ideal-world functionality to describe the security properties of a DAP scheme, which captures both the correctness and privacy of the scheme in a single intuitive definition. Moreover, as we show in later sections, this simulation-based definition can easily be augmented to incorporate policies.

Augmenting existing constructions with simulation-sound zkSNARKs. Given an improved simulation-based definition for DAP schemes, we encounter a technical problem that prevents us from proving security of the scheme: we require zero-knowledge proofs which are sound and extractable in the presence of simulated proofs, i.e., we need simulation sound proofs. However, the zk-SNARK used in Zerocash provides no such guarantees. To address this we build the first instantiation of simulation sound zk-SNARKs.

While this seems like a small step and uses folklore techniques, it is necessary and must be done carefully to ensure security and efficiency. Our approach preserves succinctness and through clever choice of primitives, has almost no effect on Zerocash performance. For space reasons, we defer further description to the full paper [10].

Anti-money laundering policies for DAP schemes. Given a stronger formal foundation, we can now explore and reason about a wide range of anti-money laundering policies that still preserve user privacy. In fact we are able to go substantially beyond this and provide policies that simultaneously allow the authorities to both trace coins as they go from individual to individual and retrieve all of a particular user's transactions and provide an accountable record of when and why those powers were used. We provide examples of several concrete compliance policies that leverage our new techniques. These include policies to enforce regulatory closure, enforced tax payments, spending limits, coin tracing, and user tracing. We also provide the first schemes in the decentralized e-cash setting that allow for coin and user tracing while maintaining an accountable record to limit abuse of those powers.

Paper outline. The remainder of this paper proceeds as follows. In Section 2 we provide definitions for secure DAP schemes and propose our modified security definition. In Section 3 we introduce the Zerocash construction and propose modifications to achieve our new definitions. In Section 4 we show how to modify a DAP scheme to enforce various policies. In Sections 5 and 6 we detail specific policies for coin tracing and accountable user tracing.

2 Decentralized Anonymous Payments

Decentralized Anonymous Payment (DAP) systems were introduced by Ben-Sasson et al. [2]. A complete formal definition can be found in [2]. A DAP system consists of an append-only ledger and a tuple of polynomial time algorithms ($\mathsf{Setup}, \mathsf{CreateAddress}, \mathsf{Mint}, \mathsf{Pour}, \mathsf{VerifyTransaction}, \mathsf{Receive}$) with the following (informal) semantics:

$\mathsf{Setup}(1^\lambda) \to params$ generates global parameters for the system.

$\mathsf{CreateAddress}(params) \to (\mathsf{addr}_{\mathsf{pk}}, \mathsf{addr}_{\mathsf{sk}})$ outputs an individual address (key pair) for each user.

$\mathsf{Mint}(params, \mathsf{addr}_{\mathsf{pk}}, v) \to (\mathsf{cm}, \pi, \mathsf{trap})$ creates a new transaction embedding a coin of value v, a proof π, and outputs a trapdoor trap for spending the coin.

$\mathsf{Pour}(params, \mathsf{cm}_1, \mathsf{cm}_2, \mathsf{trap}_1, \mathsf{trap}_2, \mathsf{addr}_{\mathsf{pk},1}, \mathsf{addr}_{\mathsf{pk},2}, v_1, v_2, v_{\mathsf{pub}}) \to (\mathsf{cm}_1, \mathsf{cm}_2, \pi)$ takes as input two coin commitments and the corresponding trapdoors, as well as a public value output and two new coin values and destination addresses, and outputs a new transaction embedding the new coins.

$\mathsf{VerifyTransaction}$ validates any of the above transactions.

Receive scans the transactions on the ledger, using the user's corresponding secret key to identify transactions addressed to the user.

Informally, a DAP scheme uses these transactions along with a trusted append-only ledger to transact funds between different users. By issuing "coins" in exchange for work, or for non-anonymous currencies, users can split, merge and combine coins of arbitrary value. Additionally, senders can transmit funds to another user without revealing the contents of the transfer to any third party.

2.1 Existing definitions and limitations

The DAP security definitions provided by Ben-Sasson et al. [2] are game based. As a result, these definitions are complex, spanning three games: one for anonymity, one for non-malleability (needed to ensure correct integration into Bitcoin), and one for balance. While non-malleability is simple, both the game for anonymity ("ledger indistinguishability") and the one preventing coin forgery ("balance") are quite complex. More importantly, these definitions are also incomplete: they do not fully enforce correctness. Unfortunately, malicious users can exploit these weaknesses to break other properties in cryptographic systems.

Specifically, in Zerocash, each Pour transaction contains one or more serial numbers related to the coins spent. These are designed to prevent double spending, and are formed as a combination of the coin recipient's private key and randomness chosen by the sender. Since the private key is fixed, if the sender uses the same randomness in two different transactions, then the serial numbers will be identical. As a result, the recipient will be unable to spend both transactions, since the duplicate serial numbers will appear to indicate a double spend. This would allow an attacker to send two payments of e.g. \$500 and \$1,000, and then ask for a refund on the "accidental" \$1,000 dollar payment. While this attack costs the attacker \$500, since both payments go through and she only gets one back, it also costs the victim \$500 , since the "accidental" payment can never be spent. This attack is easily fixed by having the recipient check for duplicate serial number randomness over all previous transactions they received.

Nonetheless, this attack shows that the "balance" definition does not fully capture what we assume about a currency system. Clearly, some part of the security definitions in a DAP scheme should prevent this, yet the definitions for Zerocash do not.

2.2 Simulation-based definition for DAP schemes

An alternative to using game-based definitions like those of [2] is to use a simulation-based definition. In this approach, we define our system in terms of an ideal functionality implemented by a trusted party T_P that plays the role that our cryptographic constructions play in the real system [12]. In the ideal-world experiment, a collection of parties interact with the trusted party according to a specific interface. In the real-world experiment, the parties interact with each other using cryptography. We now define the experiments:

Ideal-world execution $\mathtt{IDEAL}_{T_P,\mathcal{S},\Sigma}$**.** The ideal world attacker, $\mathcal{S}$, selects some subset of the $P_1, \ldots, P_n$ parties to corrupt and informs the T_P. The attacker controlled parties behave arbitrarily, the honest ones follow a set of strategies Σ. All parties then interact via messages passed to and from the T_P implementing an ideal functionality outlined in Figure 1.

- $\mathsf{addr}_{\mathsf{pk}}^{new} \leftarrow \mathsf{RegisterID}(U, \mathsf{addr}_{\mathsf{pk}})$. Registers an identity as belonging to a user.
  - The input is a user U and an address $\mathsf{addr}_{\mathsf{pk}}$. Either U or $\mathsf{addr}_{\mathsf{pk}}$ is allowed to be null but not both.
  - If $\mathsf{addr}_{\mathsf{pk}}$ is null, the trusted party T_P generates an address $\mathsf{addr}_{\mathsf{pk}}^{new}$ and stores $(U, \mathsf{addr}_{\mathsf{pk}}^{new})$. It then returns $\mathsf{addr}_{\mathsf{pk}}^{new}$ to U.
  - If U is null, the trusted party stores $(NULL, \mathsf{addr}_{\mathsf{pk}})$ and returns $NULL$.
  - If both U and $\mathsf{addr}_{\mathsf{pk}}$ are non-null, T_P updates the record for $\mathsf{addr}_{\mathsf{pk}}$ to $(U, \mathsf{addr}_{\mathsf{pk}})$. It then sets $\mathsf{addr}_{\mathsf{pk}}^{new}$ to $\mathsf{addr}_{\mathsf{pk}}$ and returns $\mathsf{addr}_{\mathsf{pk}}^{new}$.
- $\mathsf{c} \leftarrow \mathsf{Mint}(v, \mathsf{addr}_{\mathsf{pk}}, U)$. Creates a coin.
  - The input is a value v, a destination address $\mathsf{addr}_{\mathsf{pk}}$, and the user U who owns the address.
  - The trusted party T_P checks to make sure that $\mathsf{addr}_{\mathsf{pk}}$ has been registered to user U. If not, reject.
  - Else, T_P generates a unique random transaction id c and stores $(\mathsf{c}, v, \mathsf{addr}_{\mathsf{pk}}, U)$.
- $[\mathsf{c}_0^{new}, \dots, \mathsf{c}_n^{new}] \leftarrow \mathsf{Pour}([(\mathsf{c}_0, \mathsf{addr}_{\mathsf{pk},0}, v_0, U_0), \dots, (\mathsf{c}_n, \mathsf{addr}_{\mathsf{pk},n}, v_n, U_n)], v_{\mathsf{pub}})$. Generates n new coins.
  - The input is a list of tuples of (previous transaction id c_i, destination address $\mathsf{addr}_{\mathsf{pk},i}$, transaction value v_i, user U_i) and a public value v_{pub}.
  - The trusted party T_P checks to make sure that each $\mathsf{addr}_{\mathsf{pk},i}$ has been registered to U_i. If $\mathsf{addr}_{\mathsf{pk},i}$ has already been registered to another user, reject. Else, T_P updates the record for $\mathsf{addr}_{\mathsf{pk}}$ to $(U_i, \mathsf{addr}_{\mathsf{pk},i})$.
  - Else, for each input c_i, T_P retrieves each transaction and checks that it is owned by the user U_i. If not, reject.
  - T_P also checks that the sum of the values of the input transactions equals the some of the values of the output transactions plus v_{pub}.
  - The trusted party then creates n new random transaction id's c_i^{new} each with value v_i belonging to $\mathsf{addr}_{\mathsf{pk},i}$.
  - Finally, it deletes the input transaction(s) and writes the new ones to the database as $(\mathsf{c}_i^{new}, v_i, \mathsf{addr}_{\mathsf{pk},i}, U_i)$.
  - T_P returns $[\mathsf{c}_0^{new}, \dots, \mathsf{c}_n^{new}]$ to the user.

Fig. 1: Ideal Functionality. Security of a basic DAP scheme.

Real world functionality $\mathtt{REAL}_{DAP,\mathcal{A},\Sigma}$. The real world attacker $\mathcal{A}$ controls a subset of the parties $P_1, \dots, P_n$ interacting with the real DAP scheme. The honest parties execute the commands output by the strategy Σ using the DAP scheme while the attacker controlled parties can behave arbitrarily. We assume that all parties can interact with a trusted append only ledger.

For generality we present our ideal functionality definitions, as well as subsequent transaction policies, as operations on n coins in and n coins out. As we do not wish to re-present all of Zerocash, however, for brevity we deal concretely with the existing Zerocash construction which was defined for the 2-in-2-out case.

Definition 1. *We say that DAP securely emulates the ideal functionality provided by T_P if for all probabilistic polynomial-time real-world adversaries $\mathcal{A}$ and all honest party strategies Σ, there exists a simulator $\mathcal{S}$ such that for any ppt distinguishers d:*

$$P[d(\mathtt{IDEAL}_{T_P,\mathcal{S},\Sigma}(\lambda)) = 1] - P[d(\mathtt{REAL}_{DAP,\mathcal{A},\Sigma})(\lambda) = 1] \leq negl(\lambda)$$

Theorem 1. *The basic DAP scheme described in [2] and the full version of this paper satisfies Definition 2.1 given the existence of simulation-sound zkSNARKs, collision resistant hash functions, PRFs, statistically hiding and computationally binding commitments, one-time strongly-unforgeable digital signatures, and key-private public key encryption.*

See full version for a proof of Theorem 1 [10].

3 Modifying and extending Zerocash

We use the same notation for addresses, Merkle tree, ledger, and other components as Zerocash [2]. However, to meet the new definitions we must lightly modify the existing construction. In particular, we need a mechanism to allow us to extract who a Mint transaction was authored by, and we need to extend the definitions to allow for policies to be applied that validate transactions.

We modify the Zerocash construction in two ways. First, by adding a proof to the Mint algorithm that $\mathsf{cm} := \mathsf{COMM}_s(v \| \mathsf{COMM}_r(a_{\mathsf{pk}} \| \rho))$ and a verification of this proof to VerifyTransaction. This is to support our new definition. Second, we modify Receive to fix the attack described in Section 2 and check if the serial number randomness is distinct from any previously received coins, not just the ones already spent on the ledger. We refer the curious reader to the full version of the paper [10] and [2] for the complete details.

4 Policies

Given an extensible security model for Zerocash-like systems we can now consider more complex policies governing the transfer of funds than those originally embedded in Zerocash. Zerocash only enforced one restriction on funds transfer: the sum of the values in the list of output commitments must not exceed the sum of the input ones. Although that is the most basic policy governing any monetary system, far more complex ones govern our modern banking system. These include policies limiting money laundering, enforcing tax payments, and facilitating international funds transfers, among other examples. In this section, we explore several analogous policies for decentralized e-cash that still mostly preserve user privacy.

Formally, we define a policy as an algorithm that is executed each time a coin is spent ("poured"). The algorithm is parameterized by some constants (e.g., all transaction tax policies are the same except for the tax rate parameter) and takes public inputs, private inputs, and returns true or false. Policies are realized by a zero-knowledge proof, so we are essentially giving a procedural description of the standard efficiently decidable binary relation used in zero-knowledge proofs. We note that no fresh computation takes place in a policy, only validation of precomputed data.

4.1 Building blocks

To construct our policies, we use use a few basic techniques detailed here.

Adding information to coins In order to accomplish various policies, we need to store more than a numeric value inside a coin (indeed, as shown later, we will need to use coins for other things).

We do this by replacing the 192 bit zero padding in the commitment with a special *tag* used to mark what kind of counter it is. To do this we can alter the creation of a coin from

$$\mathsf{cm} := \mathsf{COMM}_s(v \| k) \quad \text{as} \quad \mathcal{H}(k \| 0^{192} \| v)$$

to

$$\mathsf{cm} := \mathsf{COMM}_s(v \| type \| data \| k) \quad \text{as} \quad \mathcal{H}(k \| type \| data \| v).$$

Policies will use the type information to ensure that an adversary cannot cause them to operate over the wrong data. Careful care must be taken so that one policy does not output data of a type used by a different policy for a different purpose. This also means that each new type requires a special Mint.

Counters A recurring problem we will encounter in these policies is the problem of how to count, e.g., how do you count how much money a user has sent, how much they need to pay taxes on, etc? Many policies depend on counting funds or transactions and as such, we treat it as a basic building block.

The approach we take is to replace the content of a coin with a counter. Upon each transaction the *policy* governing it requires the user to input their current counter state and output the new counter with the appropriate increment. Counters are a new type of "coin" and we define $\mathsf{MintCTR}$ analogously to Mint but have it reveal the *tag* in addition to v which is now called ctr. In order to maintain anonymity, these counters need to be input via the same mechanism that coin commitments are (i.e., by proving they are in a Merkle tree). In the policies below, we denote them as special inputs solely for clarity.

The particular mechanism for restricted mint depends on deployment. For certain scenarios, it may be enough to ensure that there is only one counter per address. For most scenarios where counters are tightly coupled with real world identity, we assume such mints must tied to some trusted party, probably via signature. There is thus an implicit policy on which $\mathsf{MintCTR}$s will be accepted.

We also define a utility function for verifying a counter in Algorithm 1.

```
input   : $\mathsf{CTROld}, \mathsf{CTRNew}, \mathsf{delta}, \mathsf{tag}$
output : True if counter is valid
if not $\mathsf{tag} = \mathsf{CTROld}.tag = \mathsf{CTRNew}.tag$ then
    return False;
end
return $\mathsf{CTRNew} = \mathsf{CTROld}.value + \mathsf{delta}$;
```
Algorithm 1: VerifyCounter

Identity An important aspect of any regulatory system is user *identity*. Legal concepts of identity are decidedly centralized. Since most policies are a result of legal and regulatory requirements, most of them require some notion of legal identity. In the case of counters bound to an identity, a trusted third party can issue the counters. More practically, given any trusted party who issues identity certificates, we can require a signature under its certificate chain.

Another possibility is to simply bind counters or other objects to Bitcoin/Zerocash addresses and enforce real world identity enforcement at exchanges and online wallets. In this case then, a user would be associated with a set of addresses and would later have to tie these addresses to a real-world identity. This is analogous to the current approach used in Bitcoin, where exchanges and online wallets track user identities. We believe this is the most promising avenue for real deployment.

Signatures and encryption The final building block some of our policies use are public key encryption and signature schemes. Because Zerocash uses zk-SNARKs which are proofs over arithmetic circuits, the naive approach to instantiating these schemes would be extremely costly: first build a circuit for modular exponentiation operations in some group and then build the encryption scheme on top of that. However, it is possible instead to directly calculate the signature verification operations in an extension field [9] of the arithmetic circuit, vastly increasing efficiency.

4.2 Regulatory closure

This policy allows multiple regulatory environments to coexist in the same system but the units of currency under each scheme do not mix. In practice, this policy ensures that transactions do not bridge between regulatory schemes, except through an outside mechanism such as an exchange (where legal requirements can be enforced). To implement this policy, coins, instead of just including a numeric value v, now include a regulatory type. These are created on Mint. The policy ensures that either:

1. the regulatory type of all input coins and all output coins is the same
2. or the type of all output coins is marked as $\perp$.

The point of this policy is to ensure continuity of some other regulatory scheme which may or may not be enforced by zero-knowledge. For example, such a policy could ensure that coins only came from exchanges or hosted wallets in a particular jurisdiction. In the algorithm described in Algorithm 2, we allow coins to have an arbitrary regulator marking if the output coins are signed by a valid authority. This could be a bank, an exchange, or the government itself. This allows the transfer of coins from one scheme to the other.

```
PrivateInput : CoinsIn, CoinsOut, CA, sig
Constants    : inRegType,CA
output       : True if transaction is valid
if sig $\neq \perp$ then        // if tx signed by authorized party, skip policy
    return VerifySig(CA, CoinsOut, sig);
end
outRegType $\leftarrow$ inRegType;
for e $\in$ CoinsIn do
    if e.regType $\neq$ inRegType then
        outRegType $\leftarrow \perp$; // if any regType is wrong, outRegType is $\perp$
    end
end
for e $\in$ CoinsOut do      // check each new coin has the correct regtype
    if e.regType $\neq$ outRegType then
        return False;
    end
end
return True;
```

Algorithm 2: Regulatory closure

4.3 Spending limits

In the US one common form of anti-money laundering control is a Suspicious Activity Report. This report is generated by a bank when any single transaction over $10,000 USD is reported. This can be easily implemented.

The mechanism is simple: no transaction over the limit is valid unless signed by an authority. This allows the authority to force reporting for transactions over the limit. Note, we explicitly require the signature to be public. The reason is to prevent fraud on the part of the authority. Were the signature private, a bank enforcing this could undetectably allow anyone to circumvent the reporting mechanism by issuing more signatures. On the other hand, when signatures are public, we can require that the authority log (privately) all transaction details along with the transaction itself (and perhaps a proof the details are valid). See Algorithm 3. Since the signature is used once and only known to the authority, revealing it does not impact privacy.

A more sophisticated policy would place spending limit over all transactions instead of a limit on a single transaction. For this we require counters. See Algorithm 4.

```
PrivateInput : CoinsIn,CoinsOut
PublicInput  : sig
Constants    : limit
output       : True if transaction is valid
sent ← $0$;
if sig $\neq \perp$ then    // if tx signed by authorized party, skip policy
    return VerifySig(CA, CoinsOut, sig);
end
for e $\in$ CoinsOut do
    // if we are paying to an input address, it is ‘‘change’’ back
       to the user and not counted against the limit
    if e.$addr \notin \{$c.$addr | $c $\in$ CoinsIn$\}$ then
        sent ← sent + e.$value$;
    end
end
return sent $\leq$ limit;
```
Algorithm 3: Transaction limit

Modified ideal functionality and proof sketch. We now detail, informally, the modifications to the ideal functionality necessary to capture counter based spending limits and the changes necessary to the proof. The ideal functionality changes are small: we add a per user counter maintained by the ideal functionality and modify *pour* to increment the counter as appropriate and reject the transaction if the counter is over a given limit. The resulting proof change is similarly tiny: the simulator aborts if the attacker submits a real world transaction violating their counting limit or if they try and issue a new counter. The probability of the former event is negligible if the proof system is simulation sound and the second event is not allowed in either the real or ideal protocol since the attacker cannot make new counters.

4.4 Tax

Our policies also allow us to enforce taxes, such as income taxes, sales taxes, and Value Added Tax (VAT). We accomplish this by requiring a percentage payment to a tax account with each Zerocash transaction. Because Zerocash is payment based, in practice we charge taxes on the sending side of the transaction. Obviously, this cost can be passed on to the recipient.

The tax algorithm is straight forward. All outputs being sent to other parties are summed and a percentage of that amount is added to a user's tax counter. We reveal the tag to support multiple tax authorities.[1]

Actually forcing users pay taxes is a problem left to the authority. Since they know who they issued tax counters to, they can force those people to report their

[1] We note that this tax policy is ill defined if users collude to generate a transaction using MPC where the input identities are different. This can be fixed by ensuring that there is only one identity used for all input coins. However, the same MPC mechanism could be used to share one identity and never pay taxes.

```
PrivateInput : CoinsIn, CoinsOut, $\mathsf{CTRcm}^{old}$, $\mathsf{CTRcm}^{new}$
PublicInput  : epoch,sig
Constants    : limit
output       : True if transaction is valid
spent $\leftarrow 0$;
if sig $\neq \perp$ then    // if tx signed by authorized party, skip policy
    return $\mathit{VerifySig}$(CA, CoinsOut, sig);
end
if $\mathsf{CTRcm}^{new}.value \geq$ limit then      // if counter over limit, reject
    return False;
end
for e $\in$ CoinsOut do
    // if we are paying to an input address, it is ‘‘change’’ back
       to the user and not counted against the limit
    if e.$addr \notin \{$c.$addr |$c $\in$ CoinsIn$\}$ then
        spent $\leftarrow$ spent + e.$value$;
    end
end
return VerifyCounter($\mathsf{CTRcm}^{old}$,$\mathsf{CTRcm}^{new}$,
                spent,“limit” );
```
Algorithm 4: Spending limit

```
PrivateInput : CoinsIn, CoinsOut, $\mathsf{CTRcm}^{old}$, $\mathsf{CTRcm}^{new}$
Constants    : taxrate, tag
output       : True if transaction is valid
spent $\leftarrow 0$;
for e $\in$ CoinsOut do
    if e.$addr \notin$ CoinsIn then
        spent $\leftarrow$ spent + e.$value$;
    end
end
return VerifyCounter($\mathsf{CTRcm}^{old}$,$\mathsf{CTRcm}^{new}$,
                $\lfloor$spent $\cdot$ taxrate$\rfloor$,tag);
```
Algorithm 5: Tax

income. An alternate policy would be to directly pay taxes off each transaction. This would, however, leak transaction amounts to the authority, rather than just total income.

4.5 Identity escrow

```
**PrivateInput** : CoinsIn.CoinsOut, r
**PublicInput**  : escrowct
**Constants**    : CA
**output**       : True if transaction is valid
**return** escrowct = $Pkenc(\mathsf{CA}, \mathsf{CoinsIn} \| \mathsf{CoinsOut}; r)$;
```

Algorithm 6: Identity escrow

The identity escrow policy allows governments to selectively revoke the anonymity of individual transactions, using a master key held securely by some tracing authority. The implementation of this function is simple: a copy of the transaction details is encrypted under the public key of some authority. The policy validates that the encryption is correct. We note it may be possible to forgo public key encryption if every user has a pre-shared key with the tracing authority (since it is likely this would be a bank or government which users already had to register with, this is possible) and we can authenticate the key with a signature scheme if this proves more efficient.

5 Coin tracing

We now detail how to trace individual coins.

5.1 Construction

In a coin tracing scheme, individual coins can be marked for tracing. All subsequent coins resulting from transactions on those initial coins will themselves be traceable. To implement this in Algorithm 7, each coin commitment contains a fresh key for a public key encryption scheme. The idea is that all of the information needed to trace the output coins (including the new private keys) is encrypted under the existing key for the input coin commitments. Thus, if the inputs are traced, then the outputs are traceable. If we did only this, however, the sender would be able to trace the coins because he generated the fresh keys for the outputs. Thus, we encrypt all of this output under the authority's public key. For space efficiency, instead of encrypting everything under each input key, we encrypt the same symmetric key under each public key and encrypt everything under that key.

Our tracing scheme has two major limitations: first unless someone removes the tracing key for a coin and replaces it with a dummy key, eventually all coins in the network will be traced. This is an inherent limitation of a tracing mechanism

```
PrivateInput : CoinsIn, CoinsOut, r, pkCTs, rEnc, k, ct, rKeyGen, pk, sk
PublicInput  : escrowct
Constants    : $\mathcal{G}$
Constants    : CA
output       : True if transaction is valid
if sig $\neq \perp$ then                 // tx is from e.g a bank, skip tracing
    return $\mathit{VerifySig}$(CA, CoinsOut, sig)
end
$i, j \leftarrow 0$;
for e $\in$ CoinsOut do      // check new coins' tracing keys are correct
    if (pk[$i$], sk[$i$]) $\neq PKgen(\mathcal{G};$ rKeyGen[$i$]) $or$ e.$key \neq$ pk[$i$] then
        return False;                   // key was not generated honestly
    end
    $i \leftarrow i + 1$;
end
for $e \in$ CoinsIn do // ensure k is encrypted under each tracing key
    if pkCTs[$j$] $\neq PKEnc$(e.$key$, k; rEnc[$j$]) then
        return False ;      // symmetric key k is not encrypted under
        input coin j's tracing key
    end
    $j \leftarrow j + 1$;
end
if ct $\neq enc$(k, CoinsIn||CoinsOut||sk; rEnc) then
    return False; // new tracing private keys not encrypted under k
end
// ensure ct is encrypted under authority key
return escrowct = $PkEnc$(CA, ct||pkCTs );
```

Algorithm 7: Coin tracing

that marks all coins involved in a transaction as tainted. Of course, we could have more limited versions where we map the taint bits from input coins to output coins directly (e.g., the largest input's taint bit maps to the largest output) but these lead to circumvention issues.

The second issue is that the user and the authority can collude to trace coins even if the input coins were not marked as traced. This is not an inherent limitation but rather arises from the fact that the authority always knows the decryption keys for the outer layer of encryption and the user always knows the inner layer keys.

Because of these limitations we anticipate that in a real system, coins will frequently return to exchanges and online wallets and at these locations tracing can be removed or added. To facilitate this we allow exchanges who have an authorized signing key to either add or remove tracing by changing the key associated with a coin. We note that when an exchange removes tracing this can be done in a verifiable manner by using a known dummy key as the coin key. Looking forward, we can use similar techniques to accountable user tracing in the next section to allow the exchange to accountably add or remove tracing by providing a randomized tracing key that is either the authority's key or a dummy key.

Given the above policy, the algorithm to actually do tracing is detailed in Algorithm 8.

```
input  : The list of trace ciphertexts TraceCTs, the authority's decryption key sk,
         the initial coin tracing private key ck for the coin we want to trace
output : The trace for a particular coin Trace
for e ∈ TraceCTs do     // decrypt the outer layer of all transactions
    TraceRecords.append(PKDec(sk,TraceCTs));
end
KeysToTrace.append(ck);
for ck ∈ KeysToTrace do
    // For every public key ciphertext in every pour, we attempt to
       decrypt with the target key. On success, we get back the
       symmetric ciphertext associated with the pour, and the
       decrypted symmetric key.
    encrec, k ← $FindTxByTrialDecryption$(TraceRecords, ck);
    if encrec $\neq \perp$ then
        rec ← $SymDec$(k, encrec);                  // now decrypt the record
        Trace.append(rec);       // add the decrypted record to the trace
        for $i \in \{0, 1, \ldots,$ rec.$CoinsIn.Size() - 1\}$ do
            KeysToTrace.append(rec.sk[$i$]);  // enqueue the new tracing keys
        end
    end
end
return Trace;
```

Algorithm 8: TraceCoin

The algorithm to trace coins takes an initial target coin commitment, the initial public key for that coin commitment, and the key to decrypt all tracing records. It simply searches for all coin commitments output by that transaction, maps the new keys to those coin, decrypts the record with the existing keys, and adds those coin commitments to the list of ones to search for.

5.2 Security

Modified ideal functionality To allow tracing, we need to modify the Ideal Functionality to (1) have a special party designated the tracing authority, (2) allow coins to be marked as traced by that authority, (3) mark all output coins as traced if any of the inputs are traced and (4) allow a report on all traced coins to be made to that authority. Of course, since we allow static corruption, the tracing authority may be the adversary. We detail the modifications in Figure 2.

– $[\mathsf{c}_0^{new}, \ldots, \mathsf{c}_n^{new}] \leftarrow \mathsf{Pour}([(\mathsf{c}_0, \mathsf{addr}_{\mathsf{pk},0}, v_0, U_0), \ldots, (\mathsf{c}_n, \mathsf{addr}_{\mathsf{pk},n}, v_n, U_n)], v_{\mathsf{pub}}, aux)$.
  - Do steps one through six of Pour in Figure 1, adding an additional stored field (initially set to false) to denote if a coin is being traced or not.
  - If any coin c_i is marked as traced, mark all output coins as traced.
  - T_P returns $[\mathsf{c}_0^{new}, \ldots, \mathsf{c}_n^{new}]$ to the user.

– $\mathsf{Trace}(\mathsf{c}, U)$
  - The input is a coin c and a requesting party U.
  - If the requesting party U is the tracing authority, update the stored value for c and mark it as traced. Else, reject.

– $[\mathsf{c}_0, \ldots, \mathsf{c}_m] \leftarrow \mathsf{Report}(U)$
  - The T_P returns all entries $[\mathsf{c}_0, \ldots, \mathsf{c}_m]$ for coins marked as traced if the requesting party U is the tracing authority, otherwise return $\perp$.

Fig. 2: Modified ideal functionality for coin tracing.

Proof sketch If the tracing authority is not corrupted, then the simulator can always use random values for the escrow ciphertext and the probability of abort is bounded by the security of the outer encryption scheme.

We now deal with the more complex case: a corrupted tracing authority who the simulator must generate notifications for. There are two subcases: traced coins and non-traced coins. For traced coins, the probability that the simulator fails in decrypting both layers of the tracing ciphertext (and hence cannot mark the coins appropriately in the ideal functionality) is bounded by the soundness of the proof system. For non-traced coins in the ideal model, the simulator can use a random ciphertext for the inner symmetric ciphertext *ct* and the associated encryptions of its key. For untraced coins, since the real world tracing authority does not have the keys for non-traced coins, the probability of abort is bounded by the security of the public key encryption scheme.

6 Accountable user tracing

```
PrivateInput : CoinsIn, CoinsOut, pk, sig, r
PublicInput  : usertracect
Constants    : CA
output       : True if transaction is valid
if $VerifySig(\mathsf{CA}, \mathsf{pk}, \mathsf{sig})$ then
    return $\mathsf{usertracect} = Pkenc(\mathsf{pk}, \mathsf{CoinsIn} \| \mathsf{CoinsOut}; r)$;
else
    return False;
end
```

Algorithm 9: Accountable user tracing

The problem with the original escrow above is that it provides no privacy from the tracing authority: all users' data is encrypted under a single key the authority holds. The only privacy guarantees users have are based on trust. And even those are limited: in order to find the coins used by a specified user, the authority needs to decrypt all coins. Ideally, we would have a system that not only allows the authority to only open coins of traced users, it creates an accountable record that they did so.

To do so, we borrow an idea from Accountable Tracing Signatures [14]: each user encrypts their data under a unique key they are given by the authority. If they are being traced, this key is a randomized version of the authority's public key. If, as is the common case, users are not being traced, the key is a randomized version of a null key (e.g. a random group element). Crucially, without knowledge of the randomness, users cannot distinguish which key they are getting at the time, but if the authority later reveals the randomness they can provably tell they were or were not traced. Thus we term this system accountable user tracing.

This requires one small change to the standard identity escrow policy: we have to allow an arbitrary key and ensure that it is signed by the authority and for the current epoch.

The larger question is key management. If we simply require the authority to reveal who they were searching a year later, this is simple: each user gets a key every time period (e.g. daily) and by the disclosure deadline for searches, the authority reveals the randomness used to make the key.

While appealing theoretically and both efficient and simple, this is unfortunately not practical: given the prevalence of National Security Letters and gag orders, it is likely there are cases where the authority will never want to reveal who they searched. The standard solution to this is a transparency report which details what fraction of users were searched or offers a loose bound on the number of searches.

However, building a cryptographically binding transparency report over millions of users efficiently is somewhat problematic given that it would contain millions of records. There are various ways this might be done (e.g. techniques for batched proofs). However, we note a simple one here: use a separate instance of Zerocash to create tokens that allow search. Each user gets their key along with a proof that their key is either randomized from the null key or randomized from the authority's key and a zerocoin is spent. At the end of each time period, the authority can spend all unused coins, thus revealing what fraction of people they searched.

7 Related work

7.1 Anti-money laundering for centralized e-cash

Many, if not all of the policies in this paper have been implemented in centralized e-cash schemes, be it spending limits [7] or user and coin tracing. Somewhat surprisingly accountable user and coin tracing exists for centralized schemes [16, 17]. However none of these have been done in the decentralized setting.

Moreover, the known techniques for coin and user tracing [16, 17] require coins to be withdrawn from the authority and then deposited back and have only one transfer, whereas our techniques allow continual transactions with the need to interact with the authority only dictated by the allowable delay in starting a trace.

7.2 Hawk

In concurrent work Kosba et al. [15] present Hawk, a system for privacy preserving smart contracts with an ideal functionality based definition. Hawk's security is set in the UC model [8], which is a stronger setting than we explore here. Achieving UC security requires all proofs to include an encrypted copy of their witness and a proof of its validity. This entails two costs: first, the increase in proving cost due to circuits handling encryption and second, the loss of succinctness. In the case of Zerocash, this results in at least an 800% increase in the size of the proof due to the need to encrypt the full 64 element Merkle tree path.[2] This trade off between security and performance does not appear to affect our actual constructions here: if built using a UC security proof scheme, they should be UC secure.

8 Acknowledgements

This work was supported by: The National Science Foundation under awards EFRI-1441209 and CNS-1414023; Google ATAP; The Mozilla Foundation; and the Office of Naval Research under contract N00014-14-1-0333.

References

1. Barber, S., Boyen, X., Shi, E., Uzun, E.: Bitter to better - how to make Bitcoin a better currency. In: Proceedings of the 16th International Conference on Financial Cryptography and Data Security. pp. 399–414. FC '12 (2012)
2. Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: Security and Privacy (SP), 2014 IEEE Symposium on. pp. 459–474. IEEE (2014)
3. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: SNARKs for C: verifying program executions succinctly and in zero knowledge. In: Proceedings of the 33rd Annual International Cryptology Conference. pp. 90–108. CRYPTO '13 (2013)
4. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Succinct non-interactive zero knowledge for a von Neumann architecture. In: Proceedings of the 23rd USENIX Security Symposium. Security '14 (2014), available at `http://eprint.iacr.org/2013/879`
5. Biryukov, A., Khovratovich, D., Pustogarov, I.: Deanonymisation of clients in bitcoin p2p network. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. pp. 15–29. ACM (2014)

[2] It may be possible to reduce by introducing zero-knowledge proofs that are only partially extractable. After all, the exact Merkle tree path is irrelevant to the simulator.

6. Bitansky, N., Chiesa, A., Ishai, Y., Ostrovsky, R., Paneth, O.: Succinct non-interactive arguments via linear interactive proofs. In: Proceedings of the 10th Theory of Cryptography Conference. pp. 315–333. TCC '13 (2013)
7. Camenisch, J., Hohenberger, S., Lysyanskaya, A.: Balancing accountability and privacy using e-cash. In: Security and Cryptography for Networks, pp. 141–155. Springer (2006)
8. Canetti, R.: Universally Composable Security: A new paradigm for cryptographic protocols. In: FOCS '01. p. 136. IEEE Computer Society (2001), `http://eprint.iacr.org/2000/067`
9. Danezis, G., Fournet, C., Kohlweiss, M., Parno, B.: Pinocchio coin: building zerocoin from a succinct pairing-based proof system. In: Proceedings of the First ACM workshop on Language support for privacy-enhancing technologies. pp. 27–30. ACM (2013)
10. Garman, C., Green, M., Miers, I.: Accountable privacy for decentralized anonymous payments. Cryptology ePrint Archive, Report 2016/XXX. http://eprint.iacr.org (2016)
11. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs. In: Proceedings of the 32nd Annual International Conference on Theory and Application of Cryptographic Techniques. pp. 626–645. EUROCRYPT '13 (2013)
12. Goldreich, O.: Foundations of cryptography: volume 2, basic applications. Cambridge university press (2004)
13. Groth, J.: Short pairing-based non-interactive zero-knowledge arguments. In: Proceedings of the 16th International Conference on the Theory and Application of Cryptology and Information Security. pp. 321–340. ASIACRYPT '10 (2010)
14. Kohlweiss, M., Miers, I.: Accountable tracing signatures. Cryptology ePrint Archive, Report 2014/824 (2014), `http://eprint.iacr.org/`
15. Kosba, A., Miller, A., Shi, E., Wen, Z., Papamanthou, C.: Hawk: The blockchain model of cryptography and privacy-preserving smart contracts (2015)
16. Kügler, D., Vogt, H.: Auditable tracing with unconditional anonymity. In: International Workshop on Information Security Application – WISA 2001. pp. 151–163 (2001)
17. Kügler, D., Vogt, H.: Offline payments with auditable tracing. In: Financial cryptography. pp. 269–281. Springer (2003)
18. Lipmaa, H.: Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In: Proceedings of the 9th Theory of Cryptography Conference on Theory of Cryptography. pp. 169–189. TCC '12 (2012)
19. Lipmaa, H.: Succinct non-interactive zero knowledge arguments from span programs and linear error-correcting codes. In: Proceedings of the 19th International Conference on the Theory and Application of Cryptology and Information Security. pp. 41–60. ASIACRYPT '13 (2013)
20. Miers, I., Garman, C., Green, M., Rubin, A.D.: Zerocoin: Anonymous distributed e-cash from bitcoin. In: Security and Privacy (SP), 2013 IEEE Symposium on. pp. 397–411. IEEE (2013)
21. Parno, B., Gentry, C., Howell, J., Raykova, M.: Pinocchio: nearly practical verifiable computation. In: Proceedings of the 34th IEEE Symposium on Security and Privacy. pp. 238–252. Oakland '13 (2013)
22. Reid, F., Martin, H.: An analysis of anonymity in the Bitcoin system. In: Proceedings of the 3rd IEEE International Conference on Privacy, Security, Risk and Trust and on Social Computing. pp. 1318–1326. SocialCom/PASSAT '11 (2011)