

Osman Biçer and Alptekin Küpçü

# Anonymous, Attribute Based, Decentralized, Secure, and Fair e-Donation

**Abstract:** E-cash and cryptocurrency schemes have been a focus of applied cryptography for a long time. However, we acknowledge the continuing need for a cryptographic protocol that provides global scale, decentralized, secure, and fair delivery of donations. Such a protocol would replace central trusted entities (e.g., charity organizations) and guarantee the privacy of the involved parties (i.e., donors and recipients of the donations). In this work, we target this online donation problem and propose a practical solution for it. First, we propose a novel decentralized e-donation framework, along with its operational components and security definitions. Our framework relies on a public ledger that can be realized via a distributed blockchain. Second, we instantiate our e-donation framework with a practical scheme employing privacy-preserving cryptocurrencies and attribute-based signatures. Third, we provide implementation results showing that our operations have feasible computation and communication costs. Finally, we prove the security of our e-donation scheme via formal reductions to the security of the underlying primitives.

**Keywords:** anonymous donation, attribute based signature, blockchain, anonymous credentials

DOI 10.2478/popets-2020-0069

Received 2020-02-29; revised 2020-06-15; accepted 2020-06-16.

## 1 Introduction

Since the first secure e-cash scheme was proposed by Chaum [21], digital currencies have been a center of attention in applied cryptography. One development of practical importance has been the invention of Bitcoin [54], which combines the ideas of conventional e-cash, proof of work [25], distributed systems, and game theory for a decentralized electronic payment scheme. The emergence of Bitcoin has not only contributed to state-of-the-art cryptocurrency schemes, but also brought to attention the blockchain, which is a decentralized and unalterable public ledger. Informally, a public ledger is

an append-only bulletin board, where new data can be added only if it satisfies a predetermined criterion. The proposed Bitcoin and blockchain mechanism depends on only proof of work (PoW) [25] instead of requiring a trusted third party for maintenance. PoW can be briefly described as a race among blockchain miners for computing brute-force solutions to a cryptographic puzzle with ever-changing parameters. Recent blockchain and Bitcoin research has focused on novel applications [2, 4, 6, 8, 40, 44, 45, 58], improved security [28–30, 39], and enhanced privacy [51, 60]. In this paper, we focus on the novel problem of anonymous decentralized online donations as a public ledger application.

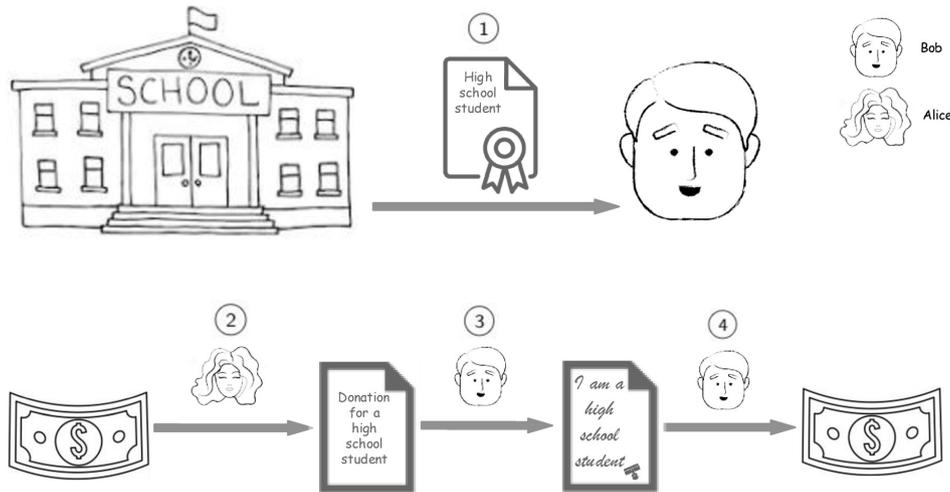
The context that we consider is similar to the “helping a stranger” described in CAF 2018 World Giving Index [36]. There, international charity organizations (ICO) match the donor and recipients based on the former’s attribute choices. The size of the charity organizations is usually underestimated. In fact, they have a grand sector of its own with numbers as high as more than 180,000 organizations only in United Kingdom [11] and more than 2 out of 3 people donate in countries like Indonesia, Australia, New Zealand, America, Ireland, United Kingdom [36]. We acknowledge this high demand for charity, and differentiate our work from this traditional setting by being *decentralized*, i.e., by eliminating ICOs to carry out the donation process. Our work enables donors to choose recipients based on their attributes in one compact system, providing fairness<sup>1</sup> of donation distributions by allowing each recipient to obtain a limited amount in each time period.

In an online donation scheme, a donor (Alice) chooses a recipient (Bob) based on his attributes without meeting him in person. Alice picks an attribute policy (e.g., the recipient should be “a high school student” AND “a person whose family’s monthly income is lower than \$750 per month”), and publishes it in a publicly visible venue (the Ledger) along with the donation amount. She might go offline afterwards. Assume that Bob meets this criteria, and has obtained attribute tokens from the authorities (e.g., student certificate from his school, and low income certificate from the ministry). Upon seeing this donation on the Ledger, Bob

**Osman Biçer:** Koç University, obicer17@ku.edu.tr

**Alptekin Küpçü:** Koç University, akupcu@ku.edu.tr

<sup>1</sup> For a further discussion on fairness, see Section 7.



**Fig. (1)** The flow of the online donation scheme that we propose. ① Bob obtains his attribute token from an attribute authority. ② Alice publishes the donation and the attribute policy on a publicly seen medium. She might go offline afterwards. ③ Upon seeing the donation, Bob claims the donation by showing the proof of his conformity to the attribute policy that he generated via the attribute token. ④ Bob then spends the donation amount, according to his wish.

proves his conformity to this attribute policy via the obtained attribute tokens to claim the donation. Figure 1 shows the flow of the online donation scheme. The identities of all the parties involved remain anonymous and their actions remain unlinkable during the whole process. Further, the protocol ensures that Bob will not be able to claim more donations (even if he satisfies their policies) than a publicly set amount from the system in a time period for fairness to the other people. The system also ensures that someone (who does not meet the attribute policy that Alice picked), cannot claim any money from her donation. We list the aspects that an e-donation scheme needs to satisfy as follows:

- **donor privacy:** it should prevent the leakage of Alice’s identity and linkage between multiple donations by her.
- **recipient privacy:** it should prevent linking between donations that Bob has received and leakage about his identity.
- **fair distribution of donations:** it should not allow Bob to receive more than a publicly determined amount of donation from the system in a given time period for fairness to other potential recipients.
- **decentralization:** it should not require trusted third parties (e.g., charity organizations) for the donation process.<sup>2</sup>

<sup>2</sup> We observe that trusted authorities are inevitable for checking recipient attributes (e.g., being a student), but they are not

- **always-on:** it should not require Alice and Bob to remain and interact online, in particular, it should allow Alice to go offline after sending her donation money and the recipient attribute policy.
- **security:** it should protect the overall balance of the involved parties.
- **scalability:** it should be supporting multiple attribute authorities for issuing attributes to multiple recipients and multiple donors, and Bob should be able to receive a donation from any geographical location as long as he satisfy its policy.
- **accountability:** donations should be binding such that Bob should indeed receive a donation if he proves his conformity to the associated policy.<sup>3</sup>
- **revocability:** it should allow revocation of attributes or users.

We note that some of these aspects may not be necessary in all cases. For example, some donors may choose not to be anonymous or may like to see the recipients of their donations. In our solution, we choose to provide the most capable and privacy-preserving solution that we could, and some aspects of our solution is presented in a modular way such that they can be removed for obtaining a more efficient but less capable solution.

required for monetary transactions or for each donation (they are used only for issuing attributes per user not per transaction).

<sup>3</sup> Without this property, the scheme would allow an unethical opportunity of data collection via fake donations, without the attacker losing any money.

	donor privacy	recipient privacy	fair dis- tribution	decentra- lization	always- on	secu- rity	scala- bility	account- ability	revoca- bility
ICO	✓*	✓*	✓*		✓*	✓*	✓*	✓*	✓*
NCC				✓	✓	✓	✓	✓	
ACC	✓	✓		✓	✓	✓	✓	✓	
PSC	✓	✓		✓	✓	✓	✓	✓	
Our scheme	✓	✓	✓	✓	✓	✓	✓	✓	✓

**Table (1)** Comparison of traditional international charity organizations (ICO), the non-anonymous cryptocurrency (NCC) schemes (e.g., Bitcoin [54] and Ethereum [13]), the anonymous cryptocurrency (ACC) schemes (e.g., Monero [52], Zerocoin [51], and Zerocash [60]), the private smart contract (PSC) schemes (e.g., Hawk [42], Arbitrum [37], Zether [12], ZEXE [10], and Enigma [1]) and our e-donation scheme (see Section 4 in terms of adequacy of e-donation requirements. ✓ denotes satisfaction of a requirement. By ✓\*, we refer the facts that the traditional ICOs can only provide donor and recipient privacy in a limited way, as they do learn the identities of the parties (but they may choose to hide this information when trusted); and that fair distribution of donations in these schemes is limited, as recipients can obtain donations from various ICOs; and that their security, being always-on, scalability, accountability, and revocability depend on the honesty of a single organization for all these tasks to be carried out.

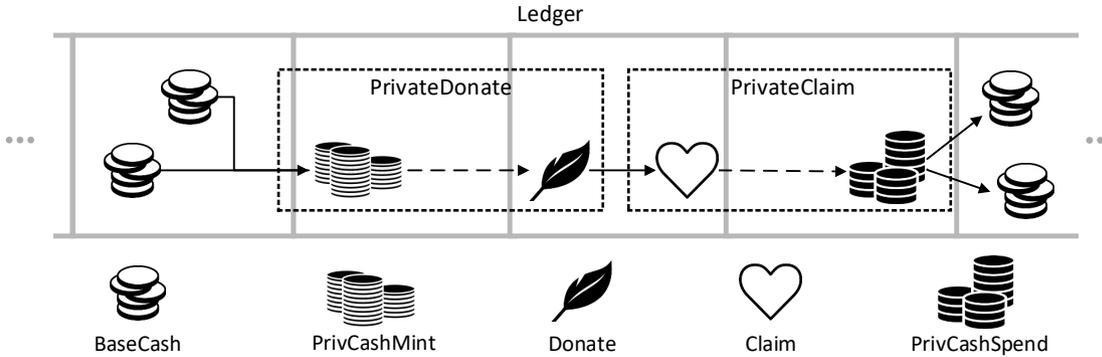
Table 1 provides a comparison of various possible solutions with respect to our e-donation requirements. We highlight that the traditional ICOs can only provide donor and recipient privacy in a limited way as they do learn the identities of the parties. Also, fair distribution of donations in this scheme is limited, as recipients may obtain donations from various ICOs. Furthermore, their security, being always-on, scalability, accountability, and revocability depend on the honesty of a single organization for all these tasks to be carried out. The motivation behind our proposed profile relies on the report of [11], showing that the donor behavior tends to be simplistic with few attribute choices in recipients and caring more about fast delivery and ease of the donation process than finding the best fitting recipient.

**System Model.** In our framework, in each state or region there is a top level authority *identity provider* (e.g., Census Bureau or State Authority) that user interacts when he joins the system to generate his user token. Thus, upon joining the system, each user has an identity attribute (e.g., national identity number or social security number). This is required for preventing Sybil attacks [61]. There are also attribute authorities for issuing attributes of users. An attribute authority can provide tokens for various attributes, and an attribute can be issued by more than one authority. We highlight that the system should prevent different users from combining their attributes. For a further discussion on possible authority architecture improvements, see Section 7.

We trust identity providers and attribute authorities (e.g., school districts) for honest issuing of user and attribute tokens by checking identities and eligibility of the users for attributes (e.g., high school student). The only malicious actions that authorities can take are is-

suing tokens to the undeserving users and revealing the identity of the users that applied to them for tokens, which can happen in any donation system. Yet, none of the authorities can donate or claim on behalf of a user, nor can they carry out the whole donation process as an ICO. They do not have monetary transaction abilities, and are not involved in each donation; rather, they are required only once per person while obtaining the attribute. Even the authorities collectively cannot identify a user from her transactions. Other than those potential malicious actions, no authority can deduce a user’s secret key or sign on behalf her. Besides, even the authorities cannot identify a user from her signature.

**Failed Approaches.** One might imagine a trivial online solution to the secure donation problem as follows. First, the donor publishes the attribute policy that she prefers the recipient to conform to. Next, a potential recipient and the donor together run an anonymous credential protocol (e.g., [18]), where the recipient proves his eligibility to the donor. Then, the donor directly pays the donation money to the recipient using an anonymous e-cash scheme to preserve privacy. However, although this scheme seems applicable, it can hardly satisfy some of the above-mentioned requirements, namely, online decentralization (due to the need for a trusted third party in the e-cash scheme), scalability (since it is not easy for every donor to always have secure access to all valid authority certificates and potential recipients), revocability (again, since secure access to all revocation lists or keys may be hard for every donor), and accountability (since once the recipient proves his conformity, the donor can still choose not to send the e-cash). Moreover, this trivial solution requires the recipients and the donors to remain online and to interact directly, not allowing decentralization in its full sense as described



**Fig. (2)** The flow of transactions in our e-donation scheme. The PrivCashMint and Donate transactions are made by the donor (together called PrivateDonate), while the Claim and PrivCashSpend transactions are made by the recipient of the donation (together called PrivateClaim). The BaseCash tuples referenced by the PrivCashMint transaction (leftmost ones) belong to the donor, whereas the output of the PrivCashSpend transaction belongs to the final payee(s). The straight arrows represent direct referencing of the transaction (linkable), while the dashed arrows represent the use of the outputs (unlinkable). We note that the block boxes are placed arbitrarily.

above. Lastly, in such a setting, enforcing fair distribution seems impossible. In our solution, we employ a distributed ledger together with proper novel cryptographic tools to deal with these issues, and provide the first definition and instantiation for the anonymous fair e-donation paradigm.

The anonymous decentralized e-cash schemes [51, 60] cannot be directly applied for the e-donation purpose, as they do not provide a mechanism for anonymous attribute-based money transfer between donors and recipients who are non-privy to each other. Also, it is non-trivial to directly combine these with the existing anonymous attribute proving techniques (e.g., anonymous credentials [14, 15, 17, 18, 23, 31, 48]), attribute based signatures [19, 26, 32, 49, 56, 67], group signatures [5, 16, 27, 35, 55, 62, 71]), while satisfying the above-mentioned requirements. We note that even letting ICOs use anonymous cryptocurrencies (e.g., Zerocash [60]) does not help with all of our e-donation requirements. While such a trivial scheme may increase donor and recipient privacy, it can hardly distribute donations fairly. Further, the above-mentioned shortcomings of ICOs still prevail.

An existing private smart contract scheme (e.g., Hawk [42], Arbitrum [37], Zether [12], ZEXE [10], and Enigma [1]) would also not solve the online donation problem itself, since it does not achieve fair distribution of donations. We show that combining a decentralized e-cash scheme or a private contract scheme with an attribute based signature scheme (e.g. VABS of [7]) would achieve e-donation.

**Overview of Our Techniques.** In this work, we employ a public ledger for decentralization and accountability. We highlight that according to the flowchart presented by NIST [69, Fig. 6], the use of blockchain in our application is appropriate. We highlight that although e-donation fits better to blockchain-based ledger applications, there is no limitation for using it with conventional e-cash schemes, as we abstract out the ledger implementation. The ledger can be maintained by a trusted party (e.g., the issuer of the e-cash), and all its security guarantees still hold.

Our scheme is based on three cryptographic building blocks: an underlying BaseCash scheme (i.e., our abstraction of decentralized e-cash such as Bitcoin [54]) that provides the amount of donations, an anonymized PrivCash scheme (i.e., our abstraction of decentralized anonymous e-cash such as Zerocash [60]) and an attribute-based signature scheme called VABS [7] that provides usage limitation, revocability, and decentralization features. PrivCash scheme basically has two operations, PrivCashMint for minting PrivCash (e.g., in Zerocash a Mint operation for converting the amounts in t-addresses into the ones z-addresses) and PrivCashSpend for spending minted PrivCash (e.g., in Zerocash a Pour operation by converting the amounts in z-addresses into the ones in t-addresses). A PrivateDonate transaction made by the donor consists of two main actions: one for minting some PrivCash (i.e., PrivCashMint) and one for sending it to some attribute policy (i.e., Donate) as a spending transaction. Similarly, a PrivateClaim transaction made by the recipient consists of two main actions: one for claiming a previous donation (i.e., Claim)

signed with a VABS for proving attribute policy conformity and one for spending it to a separate payee (i.e., `PrivCashSpend`). Once the donor makes a donation to an attribute policy, the donation is fixed on the Ledger as a `PrivateDonate` transaction tuple, and the donor no longer needs to be involved in its delivery. The donations on the Ledger are delivered in a first-come-first-served basis. Any qualified recipient (adhering to the attribute policy) can claim a donation with a `PrivateClaim` transaction tuple, as long as the donation (partially) remains unclaimed on the Ledger. Furthermore, for fair distribution of the donations, we limit the total amount of donations that a recipient can claim in a given time period. Figure 2 shows the flow of transactions in our e-donation scheme.

**Our Contributions.** We list the overall contributions of this paper as follows:

1. In Section 3, we propose a novel, secure, and decentralized e-donation framework. Our generic framework is ledger-based, therefore, it can be realized on top of a distributed blockchain. We scrupulously define the component algorithms and transaction tuples of the framework. We further provide formal game-based definitions for its security properties, (1) unforgeability and balance, (2) fair distribution of donations, (3) donation and recipient privacy.
2. In Section 4, we instantiate our e-donation framework with a practical and provably secure scheme. Our scheme is based on two cryptographic building blocks: an anonymized `PrivCash` scheme (i.e., our abstraction for decentralized anonymous e-cash such as `Zerocash` [60]) and a recent attribute-based signature scheme called `VABS` [7]. In Appendix A, we formally prove that our proposed scheme satisfies the security requirements of the e-donation framework, if the underlying building blocks are secure.
3. In Section 6, we provide efficiency information for our e-donation scheme, showing that our operations have reasonable computation and communication costs (i.e., for many cases, donating takes less than 1.7 min and 6 kB, while receiving donations takes around 1.7 min and less than 90 kB).

## 2 Preliminaries

**Notation.** Throughout this paper,

- $a \leftarrow B$  denotes that  $a$  gets a value from the set  $B$  sampled uniformly at random,

- $a \leftarrow B$  denotes that  $a$  gets the output of a probabilistic polynomial time algorithm (PPT)  $B$ ,
- $a := b$  denotes that  $a$  gets the value of  $b$ ,
- $A(a) \rightarrow b$  denotes that  $A$  is a PPT taking as input  $a$  and outputting  $b$  ( $A$  may output values non-deterministically), and
- $A\{B_1(b_1), B_2(b_2)\} \rightarrow (c_1), (c_2)$  denotes that  $B_1$  and  $B_2$  are two parties executing the protocol  $A$  on their inputs  $b_1$  and  $b_2$ , respectively, and  $A$  outputs  $c_1$  and  $c_2$  to parties  $B_1$  and  $B_2$ , respectively ( $A$  may output values non-deterministically to both parties).

**Ideal Public Ledger.** We now briefly describe an ideal public ledger functionality based on the public ledger functionality definition of [40]. Any party can securely read the current `LedgerC`, while only the `Ledger` writer is allowed to write on it by appending some data. Any transaction<sup>4</sup> handed to the `Ledger` writer is written to the `Ledger`, if it satisfies a validation function  $f$ , i.e., a commonly determined polynomial time function taking as input a transaction, and outputting either 1 (satisfied) or 0 (unsatisfied).

Two commonly known realizations of the `Ledger` functionality are via a blockchain or via a trusted party. In [29] and [30], the authors show that the Bitcoin protocol provides the requirements of a public ledger, based on the assumptions that the number of miners are constant, and that the majority of the hashing power belongs to the honest miners. However, existing blockchains, in practice, have issues such as connectivity problems or Eclipse attacks on the peer-to-peer network of blockchain. Therefore, when realizing the `Ledger` functionality as a blockchain, such issues should be considered. Further, blockchains do not offer network layer anonymity themselves as required by the `Ledger` functionality, but this can be provided via an additional resource such as using the TOR network [65].

**BaseCash scheme.** Our e-donation framework requires an underlying `BaseCash` scheme that functions as the value of the donations. A `BaseCash` is a tuple  $(v, p)$ , where  $v$  is a value determining the amount and  $p$  is a public key of the owner who also has a related secret key  $s$ . We note that each `BaseCash` tuple has a unique reference number  $r$  given by the `Ledger` writer when it is written on the `Ledger`. In `Zerocash`, `BaseCash` corresponds to Bitcoin or a coin belonging to t-addresses in `Zerocash` [60]. Yet, we do not separate these tuples and refer both as `BaseCash`. We note that `BaseCash` may have setup, transaction generation, and verification al-

<sup>4</sup> We use the term “transaction” to refer to any information string that conforms to some predefined rules.

gorithms, but are abstracted out for simplicity. The key generation algorithm for  $(s, p)$  is part of a digital signature scheme that is assumed to satisfy existential unforgeability under an adaptive chosen-message attack [38].

**PrivCash scheme.** Based on the decentralized e-cash definitions of Zerocash [60], we now describe a generic decentralized anonymous payment scheme PrivCash, which we utilize for donor and recipient privacy in our e-donation scheme. Yet, our description is simplified as including three types of operations PrivCashKeyGen, PrivCashMint, and PrivCashSpend. Also, it differs from the one in [60] by inclusion of a unit BaseCash amount  $u$  of each transaction (we require this for fair distribution of donations in our e-donation scheme) and exclusion of details (e.g., Merkle root inputs to spending operations) that are not generalizable. We note that similar ideas exist in [51] or other “mixing” based currencies [9, 59, 68, 72]. At the start of the PrivCash scheme execution, the Ledger writer runs the setup algorithm, obtains and publishes the global setup parameters  $pparams$ . The operations PrivCashKeyGen, PrivCashMint, and PrivCashSpend of the PrivCash scheme are defined as follows.

$\text{PrivCashKeyGen}(pparams) \rightarrow (s, p)$ : This is executed by a user for generating the user’s private/public key pair  $(s, p)$ . The user can publish the public key  $p$ , but needs to keep  $s$  as secret.

$\text{PrivCashMint}(r, s, s, p, \text{Ledger}_C, pparams) \rightarrow (c, \text{TX}_{\text{PrivCashMint}})$ : This is executed by a user (with at least  $u$  amount of BaseCash referenced as  $r$  with the corresponding secret key  $s$ ) for hiding the coins that will be spent. It takes as input the user’s private/public key pair  $(s, p)$ , the current ledger  $\text{Ledger}_C$ , and the setup parameters  $pparams$ . The algorithm outputs the minted PrivCash  $c$  and a transaction  $\text{TX}_{\text{PrivCashMint}}$ . The user sends the output  $\text{TX}_{\text{PrivCashMint}}$  to the Ledger writer and keep  $c$  as secret spending information.

$\text{PrivCashSpend}(c, s, p, \text{info}, p, \text{Ledger}_C, pparams) \rightarrow \text{TX}_{\text{PrivCashSpend}}$ : This is executed by a user for spending her private coin. It takes as input a previously minted PrivCash  $c$ , the user’s private/public key pair  $(s, p)$ , a public information  $\text{info}$  the receiver BaseCash public key  $p$ , the current ledger  $\text{Ledger}_C$ , and the setup parameters  $pparams$ . The user sends the output  $\text{TX}_{\text{PrivCashSpend}}$  (including  $\text{info}$  as plaintext) to the Ledger writer. As a result,  $u$  amount of PrivCash is deducted from the user’s balance and added to the recipient’s one as BaseCash.

Each transaction  $\text{TX}_X$  also has a corresponding verification algorithm  $(X.\text{Vrfy})$  so that the Ledger writer can verify that they are structured correctly. The Ledger writer also keeps a serial number database to pre-

vent double-spending. We note that each PrivCashSpend transaction has a unique reference number  $r$  given by the Ledger writer when it is written on the Ledger, whereas a PrivCashMint transaction is only referenced implicitly (by spending the minted coin by it). We also note that in case Zerocash is used as PrivCash, PrivCashKeyGen, PrivCashMint and PrivCashSpend correspond to their CreateAddress, Mint, and Pour with the following slight modifications for generality, respectively.  $s$  of PrivCashMint is used for signing the output transaction of Mint for verification with the related public key in BaseCash. PrivCashSpend is adjusted so that the output address is BaseCash public key, and  $p_1$  and  $\text{info}$  of PrivCashSpend are written to  $\text{info}$  part of the Pour transaction. We assume that BaseCash tuples and PrivCash transactions existing on a single Ledger for simplicity, yet in reality they may be on separate ledgers (e.g., Bitcoin and Zerocash). We assume that PrivCash satisfies correctness, balance and non-malleability (see Appendix B), and ledger indistinguishability (see Appendix C).

**Attribute-based Signature (ABS) scheme.** For our e-donation construction we need a scheme that enables non-interactive proving of suitability to an attribute policy while keeping the signer’s identity anonymous (i.e., other than what the attribute policy reveals). Attribute-based signatures (ABS) are the generic tools that allows this. However, since we want fair distribution of donations, our design requires the number of signatures that a signer can generate in a time period to be limited. Therefore, most of the existing ABS schemes do not provide all features for our target in one package (see Section 5 for further discussion). Fortunately, a recent proposal VABS [7] provides a versatile attribute-based signature definition and construction. The versatility comes from additional features on top of standard ABS solutions: they provide decentralization, usage limitation, revocation, threshold traceability, and authority hiding. We utilize the anonymity, signature unforgeability, and signature soundness definitions in VABS [7] for our security proofs, and provide those definitions in Appendix D for completeness. We omit the traceability related definitions, as the design of [7] is modular.

The system model of [7] requires a top level authority (i.e., an *identity provider*) in each state or region for each user to interact before joining the system. Via this interaction, the user generates an identity token that will be used while obtaining each of her other attribute tokens for attribute authorities. The identity token is also used in each of her signatures. Therefore, this would protect against Sybil attacks or collusion. We acknowledge that this requires a central authority, but we only

trust this authority, for issuing only one distinct identity token for each user.

The setup operation takes place once in the system setup to obtain the parameters  $vparams$ , and can be executed as an algorithm or as a multi-party protocol.  $VABS.IdPJoin$  and  $VABS.AuthJoin$  operations are for the identity providers and attribute authorities to join the system. Similarly, the  $VABS.UserJoin$  operation enables a user to enter the system via interacting with an identity provider to generate her identity token. To obtain their attribute tokens, users should run  $VABS.AttrIssue$  with the attribute authorities.  $VABS.Sign$  and  $VABS.Verify$  algorithms are to sign documents while only revealing attributes and verify the given VABSs, respectively. During verification,  $n$ -times periodic usage limitation can be enforced. Moreover,  $VABS.UserRevoke$  and  $VABS.AttrRevoke$  operations are provided for revocation of users and their attributes by entitled authorities. Here we provide the shortened descriptions of  $VABS.Sign$  and  $VABS.Verify$  in [7]. We omit the definitions of others and refer the reader to [7] for further details related to them, as their use in our e-donation scheme is straightforward.

$VABS.Sign(m, S, \beta, \Sigma_\beta, IAP, t, J, vparams) \rightarrow \sigma$ : Being executed by a user, it takes as input a message  $m$ , a secret key  $S$ , an attribute policy  $\beta$ , an attribute token set  $\Sigma_\beta$  that proves that the owner of  $S$  conforms to the attribute policy  $\beta$ , the public key set  $IAP$  of the identity providers and authorities, the time period indicator  $t$ , a signature counter  $J$ , and the global setup parameters  $params$ . The output of the algorithm is a VABS  $\sigma$  on  $m$ . Upon execution, the user increments the counter  $J$  for verification of his next signatures.

$VABS.Verify(m, \sigma, \beta, IAP, t, SDB, vparams) \rightarrow b$ : Executed by a verifier, it takes as input a message  $m$ , a VABS  $\sigma$ , an attribute policy  $\beta$ , the public key set  $IAP$  of the identity providers and authorities, the time period indicator  $t$ , the current signature database  $SDB$ , and the global setup parameters  $vparams$ . The output of the algorithm is a bit  $b$  defined as 1, if verification succeeds. Otherwise, it is defined as 0. If the algorithm output is 1, the signature database is updated by the addition of  $\sigma$  (i.e.,  $SDB := SDB \parallel \sigma$ ).

### 3 E-Donation Framework

In this section, we define the properties and security notions of an e-donation scheme, along with the games related to them. Our framework requires a Ledger scheme

(as described in Section 2) that can be instantiated by a trusted third party or a distributed system, and is publicly available for read access. No one can directly write on the Ledger except for the honest Ledger writer. The Ledger is characterized as permanent, i.e., whatever is written on it remains unalterable afterwards. The e-donation framework is built on top of a BaseCash scheme that makes donations worthy.

#### 3.1 Operations

We define the e-donation operations,  $GlobalSetup$ ,  $IdPJoin$ ,  $AuthJoin$ ,  $UserJoin$ ,  $RecipJoin$ ,  $AttrGen$ ,  $PrivateDonate$ , and  $PrivateClaim$ , as follows.

$GlobalSetup(1^\lambda) \rightarrow (dparams)$ : This operation is run by the Ledger writer at the start of the protocol. It takes as input a unary security parameter  $1^\lambda$ , and outputs global setup parameters  $dparams$ . The Ledger writer may also update  $dparams$  using this operation. We note that if the Ledger writer is a distributed system (as in blockchain), this operation might run as a multi-party protocol where all involved parties have the same input  $1^\lambda$ , and obtain  $dparams$  at the end.

$IdPJoin(dparams) \rightarrow (is, ip)$ : This operation is run by an identity provider for generation of its private/public key pair  $(is, ip)$ .

$AuthJoin(dparams) \rightarrow (as, ap)$ : This operation is run by an attribute authority for generation of its private/public key pair  $(as, ap)$ .

$UserJoin(dparams) \rightarrow (s, p)$ : This operation is run by a user (a donor or a recipient) for generation of its private/public key pair  $s, p$ .

$RecipJoin\{User(uid, ip), IdentityProvider(uid, is, ip)\} \rightarrow (S), (\perp)$ : This operation is a two-party protocol between a user (a potential recipient) and an identity provider for generating the user's secret key  $S$  that will be used receiving donations. The identity provider has a secret key  $is$ , and both parties know the user identity  $uid$  and the identity provider public key  $ip$ . The algorithm outputs to the user the secret key  $S$  that is tied to  $uid$ .

$AttrGen\{User(uid, S, ap, \omega), AttributeAuthority(uid, as, ap, \omega)\} \rightarrow (\sigma_\omega), (\perp)$ : This operation is a two-party protocol between a user and an attribute authority for issuing the attribute token to the user. The user has a secret  $S$  obtained from  $RecipJoin$ , the authority has a secret key  $as$ , and both parties know the user identity  $uid$ , the authority public key  $ap$ , and the attribute  $\omega$  that the user wishes to get approved upon. The algorithm outputs to the user the attribute token  $\sigma_\omega$

that proves the conformity of the user with secret key  $S$  to the attribute  $\omega$ .

$\text{PrivateDonate}(r, s, \mathbf{s}, \mathbf{p}, \beta, dparams, \text{Ledger}_C) \rightarrow (\text{TX}_{\text{Donate},1}, \dots, \text{TX}_{\text{Donate},k})$ : This operation is run by a donor for casting the donation to an attribute policy. It takes as input some  $\text{BaseCash}$   $r$  with a total of at least a unit amount donation<sup>5</sup>  $u$  with the corresponding secret key  $s$ , the donor's private/public key pair  $(\mathbf{s}, \mathbf{p})$ , an attribute policy  $\beta$  about the recipient, and the system parameters  $dparams$ . The algorithm outputs a tuple  $(\text{TX}_{\text{Donate},1}, \dots, \text{TX}_{\text{Donate},k})$  of  $k$  transactions. At least one of these transactions publishes  $\beta$ . We call the transaction tuple  $(\text{TX}_{\text{Donate},1}, \dots, \text{TX}_{\text{Donate},k})$  belonging to the same  $\text{PrivateDonate}$  operation as a transaction  $\text{TX}_{\text{PrivateDonate}}$ . We note that the amount  $u$  is deducted from the donor's balance, and is added to the claimable amounts with the policy  $\beta$  as a result.

$\text{PrivateClaim}(r, S, \Sigma_\beta, IAP, \mathbf{s}, \mathbf{p}, p, dparams, \text{Ledger}_C) \rightarrow (\text{TX}_{\text{Claim},1}, \dots, \text{TX}_{\text{Claim},h})$ : This operation is run by a recipient for claiming an unclaimed donation. It takes as input a reference previous  $\text{TX}_{\text{Donate},k}$  transaction  $r$  on the current  $\text{Ledger}_C$ , the recipient's secret key  $S$  for receiving donations, an attribute token set  $\Sigma_\beta$  for proving conformity to  $\beta$  of the referenced  $\text{TX}_{\text{Donate},k}$  issued by attribute authorities, the set  $IAP$  of identity provider and attribute authority public keys, the recipient's key pair  $(\mathbf{s}, \mathbf{p})$ , the  $\text{BaseCash}$  public key  $p$  of the payee<sup>6</sup>, and the system parameters  $dparams$ . The algorithm outputs a tuple  $(\text{TX}_{\text{Claim},1}, \dots, \text{TX}_{\text{Claim},h})$  of  $h$  transactions. At least one of these transactions publishes  $p$ . We call the transaction tuple  $(\text{TX}_{\text{Claim},1}, \dots, \text{TX}_{\text{Claim},h})$  belonging to the same  $\text{PrivateClaim}$  operation as a transaction  $\text{TX}_{\text{PrivateClaim}}$ . We note that the amount  $u$  is deducted from the claimable amounts with  $\beta$ , and is added to the payee's balance as a result.

Each transaction also has a corresponding verification algorithm, and is written to the  $\text{Ledger}$  after the  $\text{Ledger}$  writer verifies it. The public outputs of  $\text{IdPJoin}$  and  $\text{AuthJoin}$  can be written to  $\text{Ledger}$  for authenticity. We enforce each  $\text{PrivateDonate}$  and each  $\text{PrivateClaim}$  to have the same unit amount  $u$  determined by the  $\text{Ledger}$  writer (as a public system parameter, for fair dis-

<sup>5</sup> We set the unit donation amount for achieving fair distribution of donations (see Section 3.2) via VABS [7]. Yet, for further improvements it can be removed. We allow the reference to have total amount more than  $u$  for possible donation fees charged by the  $\text{Ledger}$  writer.

<sup>6</sup> The payee is not the recipient of the e-donation, but one to whom the recipient pays the e-donation.  $\text{PrivateClaim}$  operation not only ensures the privacy of the recipient but also hides the final payee of a donation, who may be the same as or different from the recipient.

tribution of donations). This issue will become clearer when we present our construction. We note that some e-donation constructions may not support revocation. Therefore, we do not formalize the related operations, yet in our e-donation construction we achieve revocation via use of VABS implicitly. Also, we note that both  $k$  and  $h$  would ideally, be set as 1 for better efficiency. Yet, currently this causes issues in privacy, which will be clearer in the privacy definition in Section 3.2.

## 3.2 Security Definitions

In this section, we provide the security requirements in an e-donation scheme  $\Pi = (\text{Ledger}, \text{BaseCash}, \text{GlobalSetup}, \text{IdPJoin}, \text{AuthJoin}, \text{UserJoin}, \text{RecipJoin}, \text{AttrGen}, \text{PrivateDonate}, \text{PrivateClaim})$ . We provide our unforgeability and balance game building on top of the conventional existential unforgeability games, the Balance of [51], and the TR-NM game of [60] (see Appendix C). We propose a fair distribution of donations definition, which we prepared building upon the soundness of  $n$ -times unlinkable scheme of [17], applying that to e-donation. Our privacy notions for donation and recipient privacy follows similar ideas to the transaction privacy in the decentralized anonymous e-cash scheme [60].

**Unforgeability and Balance.** The unforgeability and balance property of an e-donation scheme ensures that each donation has a valid donor who owns some  $\text{BaseCash}$  with enough amount, and that the recipient of a donation conforms to the attribute policy chosen by the donor. Our following definition implies that any forging or imbalance attempt within an e-donation scheme will be detected with one minus negligible probability. In fact, we cover four different possibilities of dishonest attempts (i.e., posting  $\text{TX}_{\text{PrivateDonate}}$  using another party's  $\text{BaseCash}$ , posting imbalanced  $\text{TX}_{\text{PrivateDonate}}$  (or modification of one generated by an honest user as a forged donation to a different attribute policy<sup>7</sup>), posting  $\text{TX}_{\text{PrivateClaim}}$  that references  $\text{TX}_{\text{Donate},2}$  for a user that does not satisfy the related attribute policy, and posting imbalanced  $\text{TX}_{\text{PrivateClaim}}$  (or modification of one generated by an honest user as a forged claim to a different payee<sup>7</sup>) by an active adversary. Here, we combine these four attacking possibilities in a single definition, since their definitions would be overlapping if we had defined them separately. Formally, for an e-donation protocol  $\Pi$ , a PPT adversary  $\mathcal{A}$ , a challenger  $\mathcal{C}$  who also controls the  $\text{Ledger}$  writer,

<sup>7</sup> This is important in case of Blockchain use for  $\text{Ledger}$  instantiation for security against malicious network participants.

and a security parameter  $\lambda$ , consider the following unforgeability experiment  $\text{eDonForge}_{\mathcal{A}, \Pi}^{u, n, \delta}(\lambda)$ :

1.  $\mathcal{C}$  gives to  $\mathcal{A}$  the security parameter  $1^\lambda$ , the system parameters  $dparams$  obtained by running  $\text{GlobalSetup}(1^\lambda)$ , and read access to all Ledger content.  $\mathcal{A}$  is also given the unit donation amount  $u \in \text{poly}(\lambda)$ , the total donation amount  $D = n \cdot u$  that can be received by a recipient in a given time period where  $n \in \text{poly}(\lambda)$ , and the duration  $\delta$  of a time period<sup>8</sup>. Time period counter  $t$  is initialized as  $t := 1$ , and is started.
2. At any step,
  - (a)  $\mathcal{A}$  is allowed to generate identity providers and attribute authorities and issue secret keys and attributes from those authorities.  $\mathcal{A}$  shares authority public keys with  $\mathcal{C}$  so that it would be able to verify the transactions. Similarly,  $\mathcal{A}$  can ask  $\mathcal{C}$  to generate authorities by  $\text{IdPJoin}$  and  $\text{AuthJoin}$  operations, respectively. In this case,  $\mathcal{C}$  shares authority public keys with  $\mathcal{A}$ , and provides  $\text{RecipJoin}$  and  $\text{AttrGen}$  oracle access to  $\mathcal{A}$ .
  - (b)  $\mathcal{A}$  can generate users by running  $\text{UserJoin}$  and run  $\text{RecipJoin}$  with any identity provider to upgrade any of those users to a recipient. Also,  $\mathcal{A}$  can generate polynomially-many different transaction tuples of  $\text{TX}_{\text{PrivateDonate}}$  or  $\text{TX}_{\text{PrivateClaim}}$  or  $\text{BaseCash}$  employing users under its control, and give the tuples to  $\mathcal{C}$  for writing to the Ledger. Similarly,  $\mathcal{A}$  can ask  $\mathcal{C}$  to generate users by  $\text{UserJoin}$  and to upgrade any user to a recipient by  $\text{RecipJoin}$  with any identity provider. In this case their public keys are given to  $\mathcal{A}$ . In this case their public keys are given to  $\mathcal{A}$ .  $\mathcal{A}$  can also ask  $\mathcal{C}$  to generate transactions for users under  $\mathcal{C}$ 's control and write them to the Ledger. In any case, only the transactions verified by the verification algorithm are written to the Ledger.
3.  $\mathcal{C}$  generates  $\text{BaseCash}$  tuples  $(v_1, p_1), \dots, (v_\ell, p_\ell)$  where  $\ell \in \text{poly}(\lambda)$  with  $\mathcal{A}$ 's request, writes them to the Ledger, and keeps the related secret keys  $s_1, \dots, s_\ell$  private.
4.  $\mathcal{A}$  eventually returns an attribute  $\omega$  (that was not queried to  $\text{AttrGen}$  oracles) to  $\mathcal{C}$ . We note that  $\omega$  can only be verified by the (honest) authority public keys generated by the challenger. If  $\mathcal{A}$  queries an attribute authority oracle for  $\omega$ , the attribute is issued but is immediately revoked. If the scheme does

not support revocation, definition simply says that such an attribute assignment request will be denied.

5.  $\mathcal{A}$  eventually returns (at least) one of the following:
  - (1) a transaction tuple  $\text{TX}_{\text{PrivateDonate}}$  whose references include a subset of the  $\{(v_1, p_1), \dots, (v_\ell, p_\ell)\}$  (as if a malicious user is spending some honest user's coins),
  - (2) a transaction tuple  $\text{TX}_{\text{PrivateDonate}}$  without the deduction of the amount  $u$  from the donor's balance, or without the addition of  $u$  to the balance of unclaimed donations with the same policy, or with the alteration of the balance of another party or of unclaimed donations with another policy, or that can replace an existing transaction tuple  $\text{TX}'_{\text{PrivateDonate}}$  with a different policy on the Ledger generated by a user under  $\mathcal{C}$ 's control (and the verification algorithm verifies all the transactions on the Ledger upto and including  $\text{TX}_{\text{Donate}, k}$ )
  - (3) a transaction tuple  $\text{TX}_{\text{PrivateClaim}}$  that references a  $\text{TX}_{\text{Donate}, 2}$  with a policy  $\beta \wedge \omega$  for any  $\beta$  (as if claiming a donation without satisfying the attribute),
  - (4) a transaction tuple  $\text{TX}_{\text{PrivateClaim}}$  that has a final payee that receives more amount than  $u$  without the deduction of the amount  $u$  from the donations with the same policy  $\beta$  that are unclaimed, or without the addition of  $u$  to the payee's balance, or with the alteration of the balance of any other party or of unclaimed donations with another policy, or that can replace an existing transaction tuple  $\text{TX}'_{\text{PrivateClaim}}$  on the Ledger with a different payee generated by a user under  $\mathcal{C}$ 's control (and the verification algorithm verifies all the transactions on the Ledger upto and including  $\text{TX}_{\text{Claim}, k}$ ).  $\mathcal{A}$  sends the generated tuples to  $\mathcal{C}$ . The output of the game is defined as 1 (i.e.,  $\mathcal{A}$  wins), if any such transaction tuple is written to the Ledger. Otherwise, the output of the game is defined as 0 (i.e.,  $\mathcal{A}$  loses).

**Definition 1** (Unforgeability and Balance). *An e-donation scheme  $\Pi$  is unforgeable and balanced, if  $\forall u, n, \delta \in \text{poly}(\lambda)$ , for every PPT adversary  $\mathcal{A}$ , there exists a negligible function  $n(\cdot)$  such that*

$$\Pr[\text{eDonForge}_{\mathcal{A}, \Pi}^{u, n, \delta}(\lambda) = 1] < n(\lambda)$$

**Fair Distribution of Donations.** By fairness in donation distribution, we refer to the objective that none of the recipients would be allowed to claim too many donations even if she satisfies their policies. To achieve this, we limit the total donation amount that can be received by a recipient within a time interval. Our definition ensures that any adversarial attempt within an e-donation scheme for obtaining more than the allowed amount within a time period will be detected with one

<sup>8</sup> We define  $\delta$  as a quantized value, i.e., a multiple of computation steps of  $\mathcal{A}$ .

minus negligible probability. For an e-donation protocol  $\Pi$ , a PPT adversary  $\mathcal{A}$ , a challenger  $\mathcal{C}$  who also controls the Ledger writer, and a security parameter  $\lambda$ , consider the following fair distribution of donation experiment  $\text{FairDist}_{\mathcal{A},\Pi}^{u,n,\delta}(\lambda)$ :

1.  $\mathcal{C}$  gives to  $\mathcal{A}$  the security parameter  $1^\lambda$ , the system parameters  $dparams$  obtained by running  $\text{GlobalSetup}(1^\lambda)$ , and read access to all Ledger content.  $\mathcal{A}$  is also given the unit donation amount  $u \in \text{poly}(\lambda)$ , the total donation amount  $D = n \cdot u$  that can be received by a recipient in a given time period where  $n \in \text{poly}(\lambda)$ , and the duration  $\delta$  of a time period. Time period counter  $t$  is initialized as  $t := 1$ , and is started.
2. At any step,
  - (a)  $\mathcal{A}$  is allowed to generate identity providers and attribute authorities and issue secret keys and attributes from those authorities.  $\mathcal{A}$  shares authority public keys with  $\mathcal{C}$  so that it would be able to verify the transactions. Similarly,  $\mathcal{A}$  can ask  $\mathcal{C}$  to generate authorities by  $\text{IdPJoin}$  and  $\text{AuthJoin}$  operations, respectively. In this case,  $\mathcal{C}$  shares authority public keys with  $\mathcal{A}$  and provides  $\text{RecipJoin}$  and  $\text{AttrGen}$  oracle access to  $\mathcal{A}$ .
  - (b)  $\mathcal{A}$  can generate users by running  $\text{UserJoin}$  and run  $\text{RecipJoin}$  with any identity provider to upgrade any of those users to a recipient. Also,  $\mathcal{A}$  can generate polynomially-many different transaction tuples of  $\text{TX}_{\text{PrivateDonate}}$  or  $\text{TX}_{\text{PrivateClaim}}$  or  $\text{BaseCash}$  employing users under its control, and give the tuples to  $\mathcal{C}$  for writing to the Ledger. Similarly,  $\mathcal{A}$  can ask  $\mathcal{C}$  to generate users by  $\text{UserJoin}$  and to upgrade any user to a recipient by  $\text{RecipJoin}$  with any identity provider. In this case their public keys are given to  $\mathcal{A}$ .  $\mathcal{A}$  can also ask  $\mathcal{C}$  to generate transactions for users under  $\mathcal{C}$ 's control and write them to the Ledger. In any case, only the transactions verified by the verification algorithm are written to the Ledger.
3.  $\mathcal{A}$  eventually returns to  $\mathcal{C}$  a transaction tuple for  $\text{TX}_{\text{PrivateClaim},\mathcal{A}}$  whose recipient with secret key  $S$  has claimed at least  $D$  amount in the current time period. The output of the game is defined as 1 (i.e.,  $\mathcal{A}$  wins), if the transaction tuple is written to the Ledger. Otherwise, the output of the game is defined as 0 (i.e.,  $\mathcal{A}$  loses).

**Definition 2** (Fair Distribution of Donations). *An e-donation scheme  $\Pi$  provides fair distribution of donations, if  $\forall u, n, \delta \in \text{poly}(\lambda)$ , for every PPT adversary  $\mathcal{A}$ ,*

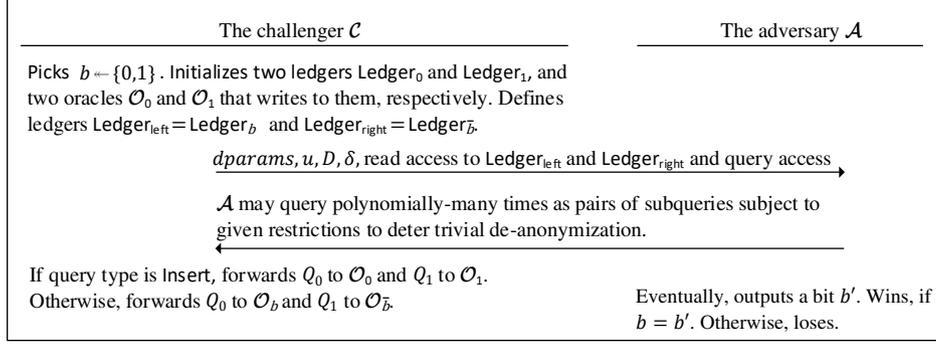
*there exists a negligible function  $n(\cdot)$  such that*

$$\Pr[\text{FairDist}_{\mathcal{A},\Pi}^{u,n,\delta}(\lambda) = 1] \leq n(\lambda)$$

**Donation and Recipient Privacy.** Our privacy notion includes hiding the identities of the donor and recipient of a donation and its final payee. We note that payee privacy is also very important for ensuring the privacy of a recipient, as the payee is likely to be recipient himself or someone with a relationship to him.

To cover the above-mentioned privacy issues, we propose a strong privacy definition similar to the ledger indistinguishability of [60]. Formally, for an e-donation protocol  $\Pi$ , a PPT adversary  $\mathcal{A}$ , a challenger  $\mathcal{C}$  who also controls the Ledger writer, and a security parameter  $\lambda$ , consider the following donation and recipient privacy-experiment  $\text{Privacy}_{\mathcal{A},\Pi}^{u,n,\delta}(\lambda)$ :

1.  $\mathcal{C}$  gives to  $\mathcal{A}$  the security parameter  $1^\lambda$ , the system parameters  $dparams$  obtained by running  $\text{GlobalSetup}(1^\lambda)$ .  $\mathcal{A}$  is also given the unit donation amount  $u \in \text{poly}(\lambda)$ , the total donation amount  $D = n \cdot u$  that can be received by a recipient in a given time period where  $n \in \text{poly}(\lambda)$ , and the duration  $\delta$  of a time period. Time period counter  $t$  is initialized as  $t := 1$ , and is started.
2. At any step,
  - (a)  $\mathcal{A}$  is allowed to generate identity providers and attribute authorities and issue secret keys and attributes from those authorities.  $\mathcal{A}$  shares authority public keys with  $\mathcal{C}$  so that it would be able to verify the transactions. Similarly,  $\mathcal{A}$  can ask  $\mathcal{C}$  to generate authorities by  $\text{IdPJoin}$  and  $\text{AuthJoin}$  operations, respectively. In this case,  $\mathcal{C}$  shares authority public keys with  $\mathcal{A}$  and provides  $\text{RecipJoin}$  and  $\text{AttrGen}$  oracle access to  $\mathcal{A}$ .
  - (b)  $\mathcal{A}$  can generate users by running  $\text{UserJoin}$  and run  $\text{RecipJoin}$  with any identity provider to upgrade any user to a recipient. Similarly,  $\mathcal{A}$  can ask  $\mathcal{C}$  to generate users by  $\text{UserJoin}$  and to upgrade any user to a recipient by  $\text{RecipJoin}$  with any identity provider. In this case their public keys are given to  $\mathcal{A}$ .
3.  $\mathcal{C}$  picks a bit  $b \leftarrow \{0, 1\}$ . It also initializes two ledger writer oracles  $\mathcal{O}_0$  and  $\mathcal{O}_1$  which writes to ledgers  $\text{Ledger}_0$  and  $\text{Ledger}_1$ , respectively.
4.  $\mathcal{A}$  is read access to the ledgers  $\text{Ledger}_{\text{left}}$  and  $\text{Ledger}_{\text{right}}$ , where  $\text{Ledger}_{\text{left}} := \text{Ledger}_b$  and  $\text{Ledger}_{\text{right}} := \text{Ledger}_{\bar{b}}$ .
5.  $\mathcal{A}$  sends its queries to  $\mathcal{C}$ . Each query  $Q$  must be in the form of a pair of subqueries  $(Q_0, Q_1)$ .  $\mathcal{C}$  forwards one of them to  $\mathcal{O}_0$  and the other one to  $\mathcal{O}_1$ . Allowed types of subqueries are  $\text{BaseCash}$ ,  $(\text{Donate}, i)$



**Fig. (3)** A high level description of the  $\text{Privacy}_{\mathcal{A}, \Pi}^{u, n, \delta}(\lambda)$  game.

for  $i \in (1, \dots, k)$ , (Claim,  $i$ ) for  $i \in (1, \dots, h)$ , and Insert. Insert queries are for the BaseCash tuples and transactions generated by the users under  $\mathcal{A}$ 's control, while the other ones are for those under  $\mathcal{C}$ 's control. The format of and the oracle's consequent action for each type should be as follows.

- (BaseCash,  $v, p$ ):  $v$  is the amount and  $p$  is the public key of the owner. If  $p$  is among the public keys of the users under  $\mathcal{C}$ 's control the oracle generates a BaseCash =  $(v, p)$ .
- (PrivateDonate,  $r, p, \beta$ ):  $r$  is the referenced BaseCash,  $p$  is the public key of a user, and  $\beta$  is an attribute policy. If no previous transaction has referenced  $r$ ,  $p$  is among the users under  $\mathcal{C}$ 's control generated by UserJoin, and  $\beta$  is a validly structured attribute policy, the oracle assigns a unique donate identifier  $qid_D$  to the query, gives  $qid_D$  to  $\mathcal{A}$ , and records the query and its  $qid_D$ .
- (Donate,  $i, qid_D$ ) for  $i \in (1, \dots, k)$ :  $qid_D$  is the referenced PrivateDonate query. If no previous transaction  $\text{TX}_{\text{Donate}, i}$  referencing  $qid_D$  is written on the oracle's Ledger, the oracle generates a  $\text{TX}_{\text{Donate}, i}$  transaction by executing the related transaction generation algorithm using the inputs of the related PrivateDonate query.
- (PrivateClaim,  $r, p, p$ ):  $r$  is the referenced  $\text{TX}_{\text{Donate}, k}$ ,  $p$  is the public key of a user, and  $p$  is the public key of the payee. If no previous transaction has referenced  $r$ ,  $p$  is among the users under  $\mathcal{C}$ 's control generated by UserJoin, and  $p$  is a validly structured public key, the oracle assigns a unique claim identifier  $qid_C$  to the query, gives  $qid_C$  to  $\mathcal{A}$ , and records the query and its  $qid_C$ .
- (Claim,  $i, qid_C$ ) for  $i \in (1, \dots, h)$ :  $qid_C$  is the referenced PrivateClaim query. If no previous

transaction  $\text{TX}_{\text{Claim}, i}$  referencing  $qid_C$  is written on the oracle's Ledger, the oracle generates a  $\text{TX}_{\text{Claim}, i}$  transaction by executing the related transaction generation algorithm using the inputs of the related PrivateClaim query.

- (Insert, TX/BaseCash) where TX is any type of transaction among  $\text{TX}_{\text{Donate}, i}$  for  $i \in (1, \dots, k)$  and  $\text{TX}_{\text{Claim}, i}$  for  $i \in (1, \dots, h)$ . If the query input includes BaseCash or the verification algorithm verifies the transaction, the oracle accepts.

Insert type of queries are for directly writing to the ledgers. Therefore, in this case, the subqueries  $Q_0$  and  $Q_1$  are forwarded to  $\mathcal{O}_b$  and  $\mathcal{O}_{\bar{b}}$  by  $\mathcal{C}$ , respectively. Other queries are for generation of the corresponding transactions. Therefore, in this case,  $\mathcal{C}$  forwards  $Q_0$  to  $\mathcal{O}_0$  and  $Q_1$  to  $\mathcal{O}_1$ . We require the following restrictions (defined similarly to *public consistency* of [60]) for each query  $Q = (Q_0, Q_1)$  to deter trivial de-anonymization.

- $Q_0$  and  $Q_1$  must be of the same type.
- If both oracles generate transactions as a result, each oracle writes the generated transaction to the corresponding ledger. Otherwise, both transactions are dropped.
- If both oracles accept as a result of Insert, each oracle writes the accepted BaseCash to the corresponding ledger. Otherwise, both transactions are dropped.
- If the type is Donate,  $i$  for  $i \in (1, \dots, k)$ , there exists two cases. If the attribute policies are being published with the resulting transactions, the policies in both referenced PrivateDonate queries by  $qid_{D, 0}$  (belonging to  $Q_0$ ) and  $qid_{D, 1}$  (belonging to  $Q_1$ ) must be the same. Otherwise, the referenced Basecash  $r$  and the related public key

- $p$  in both referenced PrivateDonate queries by  $qid_{D,0}$  and  $qid_{D,1}$  must be the same.
- If the type is Claim, $i$  for  $i \in (1, \dots, h)$ , there exists two cases. If the payee public keys are being published with the resulting transactions, the public keys in both referenced PrivateClaim queries by  $qid_{C,0}$  (belonging to  $Q_0$ ) and  $qid_{C,1}$  (belonging to  $Q_1$ ) must be the same. Otherwise, the referenced Donate, $k$   $r$  in both referenced PrivateDonate queries by  $qid_{C,0}$  and  $qid_{C,1}$  must be the same.
6. Eventually,  $\mathcal{A}$  outputs a bit  $b'$ . If  $b = b'$ , the output of the game is defined as 1 (i.e.,  $\mathcal{A}$  wins). Otherwise, the output of the game is defined as 0 (i.e.,  $\mathcal{A}$  loses).

**Definition 3** (Donation and Recipient Privacy). *An e-donation scheme  $\Pi$  provides donation and recipient privacy, if  $\forall u, n, \delta \in \text{poly}(\lambda)$ , for every PPT adversary  $\mathcal{A}$ , there exists a negligible function  $n(\cdot)$  such that*

$$\Pr[\text{Privacy}_{\mathcal{A}, \Pi}^{u, n, \delta}(\lambda) = 1] < \frac{1}{2} + n(\lambda)$$

Figure 3 provides a high level description of this game.

**Remark 1.** *Consider schemes with  $k = 1$  or  $h = 1$ . Regarding PrivateDonate, the resulting transactions need to include the referenced BaseCash and the attribute policy  $\beta$ . Having them together in one transaction directly shows whose BaseCash is donated to a particular policy. Regarding PrivateClaim, the resulting transactions need to include the referenced  $\text{TX}_{\text{Donate}, k}$  and the public key  $p$  of the payee. Having them together in one transaction would directly show which claimed donation (along with the published attribute policy) is spent to a particular payee. Therefore, in our construction, we achieve security with  $k = 2$  and  $h = 2$ , by splitting the PrivateDonate and PrivateClaim transactions into two steps, disassociating the referenced BaseCash from the attribute policy, and disassociating the referenced donation from the payee, respectively.*

## 4 Our e-Donation Scheme

In this section, we propose the first e-donation scheme  $\Pi = (\text{Ledger}, \text{GlobalSetup}, \text{BaseCash}, \text{IdPJoin}, \text{AuthJoin}, \text{UserJoin}, \text{RecipJoin}, \text{AttrGen}, \text{PrivateDonate}, \text{PrivateClaim})$  that achieves the security requirements of the generic e-donation framework, namely unforgeability and balance, fair distribution of donations, and donation and recipient privacy. Our scheme also provides multi-authority attribute issuing for recipients. We

do not propose or enforce a specific choice of the Ledger instantiation, i.e., it can be a decentralized blockchain that utilizes any consensus methodology (such as proof-of-work, proof-of-stake, or Byzantine agreement, or a combination of multiple ledgers<sup>9</sup>). Our PrivateDonate and PrivateClaim instantiation utilizes as building blocks a VABS scheme defined as in [7] and a PrivCash scheme such as Zerocash [60]. We highlight the necessity of the VABS scheme, because it is the only known ABS scheme that enables both anonymous attribute policy proving and limiting the number of signatures that a signer can generate in a time period (see Section 5 for further discussion). In what follows, we describe our e-donation scheme proposal in detail.

In GlobalSetup of our e-donation scheme, the setup operations of the BaseCash, PrivCash, and VABS schemes are executed by the Ledger writer. It also generates a BaseCash private/public key pair  $(\hat{s}, \hat{p})$  for indicating e-donation operations and completeness. The Ledger writer then sets  $dparams := (vparams, pparams, \hat{s}, \hat{p})$ . In our proposed scheme IdPJoin and AuthJoin operations are obtained by authorities via execution of VABS.IdPJoin and VABS.AuthJoin operations, respectively. Likewise, UserJoin is obtained by users via execution of PrivCashKeyGen. Further, for RecipJoin and AttrGen, the user and the authority simply run the VABS.UserJoin and VABS.AttrIssue operations, respectively. It should be noted that the VABS scheme of [7] also requires each user to run VABS.UserJoin with an identity provider, before obtaining her attribute signatures, when they first join the system.

Algorithms 1 and 2 present our PrivateDonate and PrivateClaim operations run by a donor and a recipient respectively. A PrivateDonate transaction employs PrivCashMint and PrivCashSpend transactions to maintain unlinkability, while adding the attribute policy. A PrivateClaim transaction utilizes again PrivCashMint and PrivCashSpend transactions to maintain unlinkability, and a VABS.Sign operation to prove adherence to the attribute policy. We note that the resulting  $\text{TX}_{\text{Donate}, 1}$ ,  $\text{TX}_{\text{Donate}, 2}$ ,  $\text{TX}_{\text{Claim}, 1}$ , and  $\text{TX}_{\text{Claim}, 2}$  transactions in Algorithms 1 and 2 correspond to PrivCashMint, Donate, Claim, and PrivCashSpend transactions given in Overview of Our Techniques in Section 1.

<sup>9</sup> BaseCash and PrivCash schemes, in practice, might have separate ledgers (e.g., Bitcoin and Zerocash). Here, we treat the whole system as written on a single ledger for simplicity. In case Zerocash is used for implementation, e-donation operations might be written on the ledger of Zerocash with a modification on its Ledger writer (e.g., miners' code for verifying VABS).

---

**Algorithm 1:** PrivateDonate executed by donors

---

**input:** some BaseCash  $r$  on the current Ledger<sub>C</sub> with a total amount of at least  $u$ , the related secret key  $s$  to the BaseCash, the donor's private/public key pair  $(s, p)$ , an attribute policy  $\beta$ , and the global setup parameters  $dparams = (vparams, pparams, \hat{s}, \hat{p})$   
**output:** a tuple  $TX_{PrivateDonate} = (TX_{Donate,1}, TX_{Donate,2})$

$(c, TX_{Donate,1}) \leftarrow$   
PrivCashMint( $r, s, s, p, Ledger_C, pparams$ )  
Send  $TX_{Donate,1}$  to the Ledger

$TX_{Donate,2} \leftarrow$   
PrivCashSpend( $c, s, p, \beta, \hat{p}, Ledger_C, pparams$ )  
Send  $TX_{Donate,2}$  to the Ledger

---

**Algorithm 2:** PrivateClaim executed by recipients

---

**input:** a previous unclaimed transaction  $TX_{PrivCashMint}$  referenced with  $r$  on the current Ledger<sub>C</sub>, the recipient's secret key  $S$  for receiving donations, a set of authority signatures  $\Sigma_\beta$  for proving conformity to the attribute policy  $\beta$ , the recipient's key pair  $(s, p)$ , the BaseCash public key  $p$  of the payee, the set  $IAP$  of identity provider

and attribute authority public keys, the global setup parameters  $dparams = (vparams, pparams, \hat{s}, \hat{p})$ , and a PrivateClaim counter  $J$

**output:** a tuple  $TX_{PrivateClaim} = (TX_{Claim,1}, TX_{Claim,2})$

Obtain the attribute policy  $\beta$  from donation  $r$   
Obtain the time period  $t$  from Ledger<sub>C</sub>  
 $(c, TX_{Claim,1'}) \leftarrow$   
PrivCashMint( $r, \hat{s}, s, p, Ledger_C, pparams$ )  
 $\sigma \leftarrow$  VABS.Sign( $TX_{Claim,1'}, S, \beta, \Sigma_\beta, IAP, t, J, vparams$ )  
Send  $TX_{Claim,1} := (TX_{Claim,1'}, \sigma)$  to the Ledger

$TX_{Claim,2} \leftarrow$  PrivCashSpend( $c, s, p, *, p, Ledger_C, pparams$ )  
Send  $TX_{Claim,2}$  to the Ledger

---

For verification of  $TX_{Donate,1}$ ,  $TX_{Donate,2}$ ,  $TX_{Claim,1}$ , and  $TX_{Claim,2}$  transactions, the Ledger writer runs Algorithm 3.  $SDB$  is a database on a dedicated portion of the Ledger. The Ledger writer keeps there the VABSs obtained from each Claim transaction. We highlight that some portion (e.g.,  $SDB$  can also be used) of the Ledger is dedicated to storing the identity provider and attribute authority public keys to ensure their authenticity via a public key infrastructure as in [4, 43, 70]. This would be useful as the VABS scheme [7] also allows an authority to join the system at anytime via a valid certificate that demonstrates its qualification. Furthermore, the Ledger writer can dismiss an authority from issuing some (or all) attributes by excluding its public key from verification.

---

**Algorithm 3:** Verification by the Ledger writer

---

**input:** the current Ledger<sub>C</sub> and the global setup parameters  $dparams = (vparams, pparams, \hat{s}, \hat{p})$   
**output:** updated Ledger<sub>C</sub>

On receiving a  $TX_{Donate,1}$  transaction:

if PrivCashMint.Vrfy( $TX_{Donate,1}$ ) = 1 then  
Ledger<sub>C</sub> := Ledger<sub>C</sub> ||  $TX_{Donate,1}$

On receiving a  $TX_{Donate,2}$  transaction:

if PrivCashSpend.Vrfy( $TX_{Donate,2}$ ) = 1 then  
Ledger<sub>C</sub> := Ledger<sub>C</sub> ||  $TX_{Donate,2}$

On receiving a  $TX_{Claim,1}$  transaction:

Parse  $(TX_{Claim,1'}, \sigma) := TX_{Claim,1}$   
if PrivCashMint.Vrfy( $TX_{Claim,1'}$ ) = 1 and VABS.Verify( $TX_{Claim,1'}, \sigma, \beta, IAP, t, SDB, vparams$ ) = 1 then  
Ledger<sub>C</sub> := Ledger<sub>C</sub> ||  $TX_{Claim,1}$   
 $SDB := SDB || \sigma$

On receiving a  $TX_{Claim,2}$  transaction:

if PrivCashSpend.Vrfy( $TX_{Claim,2}$ ) = 1 then  
Ledger<sub>C</sub> := Ledger<sub>C</sub> ||  $TX_{Claim,2}$

---

**Theorem 1.** *If the underlying PrivCash satisfies the PrivCash balance and non-malleability (see Appendix B), and the ledger indistinguishability [60] (see Appendix C) properties, the digital signature scheme utilized for signing PrivCashMint is existentially unforgeable under an adaptive chosen-message attack [38], and the underlying VABS protocol satisfies anonymity, signature unforgeability, and signature soundness all defined in [7] (see Appendix D); then our e-donation scheme is secure (i.e., it provides unforgeability and balance, fair distribution of donations, and donation and recipient privacy).*

*Proof Intuition.* We reduce (1) the unforgeability and balance of our e-donation scheme to the balance and non-malleability of the underlying PrivCash, existential unforgeability of the digital signature scheme under an adaptive chosen-message attack, signature unforgeability and dynamic revocation of the underlying VABS, (2) the fair distribution of donations in our e-donation scheme to the signature soundness of the underlying VABS, (3) the donation and recipient privacy of our e-donation scheme to the ledger indistinguishability [60] of the underlying PrivCash and the anonymity of underlying VABS. Full security proofs are in the Appendix.  $\square$

## 5 Related Work

**Blockchain and Cryptocurrencies.** Inspired by Bitcoin [54], many blockchain based cryptocurrencies (e.g., Ethereum [13], Litecoin [47]) are proposed and imple-

mented. For blockchain or Ledger instantiation, different consensus protocols (such as proof of work [25], proof of stake [41], proof of validation [33], or Byzantine fault tolerance [46]) would also be used. Furthermore, blockchain is not the only known decentralization method in cryptocurrencies, e.g., IOTA [57] using “The Tangle” (based on a directed acyclic graph), Phantom [64] and Spectre [63] using “blockDAG” (i.e., a directed acyclic graph of blocks). One may employ all these different techniques for Ledger instantiation.

**Transaction Privacy.** For PrivCash instantiation, many Bitcoin-like solutions depending only on pseudonyms for user privacy and not preventing linkability between transactions are not usable. Zerocash [60] can be utilized to instantiate PrivCash. For utilization of Zerocoin [51] as a PrivCash scheme, it needs to be shown to be or further improved for satisfying the balance and transaction malleability (see Appendix B), and ledger indistinguishability of (see Appendix C). Further, mixing based schemes (e.g., CryptoNote [68], Mixcoin [9], CoinShuffle [59], CoinParty [72]) also provide transaction privacy. Nevertheless, an analysis [53] on Monero [52] (i.e., a CryptoNote based cryptocurrency) shows that these type of cryptocurrencies are vulnerable to “chain-reaction analysis”. To completely avoid these type of attacks, the anonymized transactions in these protocols should use all of the previously blinded coins during mixing, which decreases their efficiency due to zero-knowledge proofs. Also, without this, these schemes do not satisfy the L-IND definition (in Appendix C) that we expect PrivCash to satisfy. Private smart contract schemes (e.g., Hawk [42], Arbitrum [37], Zether [12], ZEXE [10], and Enigma [1]) enable smart contract computations on private data. There exists no restriction for their utilization as PrivCash basis.

**Limited Times Anonymous Attribute Policy Authentication.** Our e-donation scheme essentially requires a non-interactive anonymous, multi-authority, and revocable attribute policy signature protocol, where the number of signatures in a period can be limited. We briefly mention the candidate solutions in the literature. A line of work that can be useful is anonymous credential schemes [3, 14, 15, 17, 18, 23, 31, 48] that can be utilized for proving conformation to some attribute policy while keeping the anonymity. However, they cannot be trivially modified for directly providing all requirements of the e-donation at once (e.g., fair distribution of donations among recipients). Group signature schemes [5, 16, 27, 35, 55, 62, 71] may seem as a reasonable place to start. Some group signature schemes allow revocability [27, 55]. On the other hand, again, the use

of keys in those schemes cannot be easily bounded by a fixed value  $n$ . Attribute based signature (ABS) schemes [7, 19, 24, 26, 32, 49, 56, 67] are non-interactive signature solutions for anonymous attribute policy proving. They provide binary attribute policy authentication (including Boolean relations). In particular, the existing multi-authority and decentralized ABS schemes of [7, 19, 24, 26, 56] might be modified to achieve periodic  $n$ -times usability and revocability.

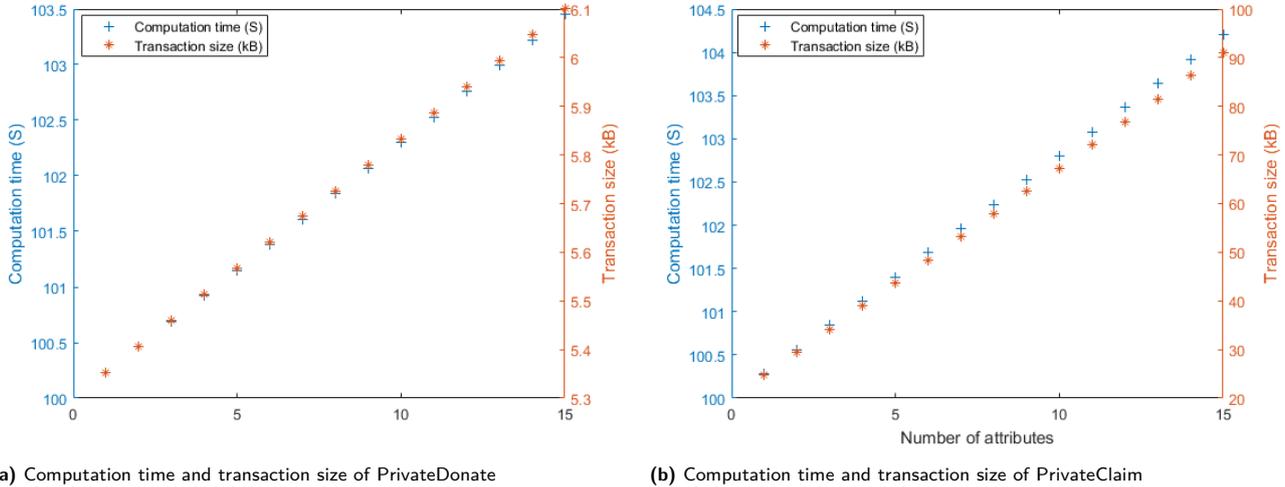
**Gitcoin**<sup>10</sup>. Gitcoin is a system to fund open-source software projects via cryptocurrencies (e.g., Ethereum [13]). It allows a developer to join the system by showing only her skills, while all parties remain anonymous. Although the flow of Gitcoin is similar to e-donation, apart from targeted audience (developers vs. general), their operational differences are as follows: (1) The former requires a centrally managed issue advertisement and accountability system while in the latter, the whole system achieve these in a decentralized way. (2) For skills, the former does not require proofs from authorities, while the latter require these proofs for attributes. (3) In the former, there exists no fairness in distribution of salaries, while the latter ensures fair distribution of donations. Although the matters (2) and (3) with Gitcoin can be resolved via updates, the matter (1) with it (i.e., system being centralized) would not be trivially improved.

## 6 Efficiency

We conclude with the efficiency analysis of our solution based on the state-of-the-art PrivCash scheme Zerocash Sapling [34] and the VABS scheme [7].

**Test Setup.** We have run our tests using the given implementation of [22] for “Output” and “Spend” transactions described in [34]. We note that the former directly corresponds to PrivCashMint, while the latter corresponds to PrivCashSpend. For our Claim operation, we have tested a prototype code for Sign and Verify algorithms of [7] via the given implemented primitives in the Cashlib cryptographic library [20, 50]. We have run our test on a machine with Intel(R) Core(TM) i7-7600U CPU @ 2.80GHz 2.90 - dualcore and 8GB RAM. The RSA and discrete logarithm moduli are set as 2048 bits, and SHA-256 is used as the random oracle. Our tests are repeated 10 times and results are averaged. The results are combined to obtain an overall cost output for our PrivateDonate and PrivateClaim operations.

<sup>10</sup> <https://medium.com/gitcoin>



(a) Computation time and transaction size of PrivateDonate

(b) Computation time and transaction size of PrivateClaim

**Fig. (4)** Computation time and total transaction size estimates for (a) PrivateDonate and (b) PrivateClaim transactions based on the number of referenced BaseCash tuples and the number of attributes in the referenced policy, respectively.

**Cost of PrivateDonate.** Figure 4a provides cost estimates for PrivateDonate operations with varying number of BaseCash tuples referenced (on which the cost linearly depends). We highlight that PrivateDonate operation is efficient enough in large majority of practical cases, since according to statistics of Nonprofits Source<sup>11</sup>, the average monthly online donation of United States of America citizens is \$52 in 2017, corresponding to 2 Zerocashes referenced as of March 2020, taking around 1.7 min and 5.4 kB. We note that the overhead is mainly due to costly operations in Zerocash.

**Cost of PrivateClaim.** Figure 4b provides cost estimates for PrivateClaim operations with varying number of attributes to be proven (on which the cost linearly depends). Indeed, while the attribute-based signature cost scales linearly with the number of attributes, we do not see this behavior in computation cost clearly, since VABS takes a few hundred milliseconds while Zerocash takes more than a hundred seconds. We highlight that the PrivateClaim operation is efficient enough in large majority of practical cases, since the donor behaviour tends to be simplistic with usually less than 5 attributes [11], taking around 1.7 min and 45 kB. The overhead in size compared to PrivateDonate is due to the included VABS.

## 7 Conclusion and Future Work

In this work, we proposed a novel e-donation framework, where donors donate to recipients by only selecting them via their attributes without meeting in per-

son. Our framework requires unforgeability and balance, fair distribution of donations, and donation and recipient privacy. Then, we constructed the first e-donation scheme that provably satisfies the requirements of the framework. We now propose some future directions for improvement of the e-donation framework and protocol.

**Fairness.** Our notion, fair distribution of donations, is based on equality of receivable donations by each user. Our work leaves improved definitions of fairness (e.g., adjustments based on location) as future work. We acknowledge that complete achievement of fairness requires replacing all ICOs with an e-donation system. This is unlikely to occur all at once in the near future and likely to require a gradual transition period.

**Attribute Authorities.** In this work, as a design choice, we do not distribute the task of issuing an attribute (e.g., threshold issuing) among multiple authorities due to the trade-off of the added benefit vs. requiring a user to have multiple interactions for each attribute (resulting in lower efficiency), and to reveal his identity to multiple authorities (increasing the risk of identity disclosure). We leave as future work design of e-donation frameworks and schemes that do not rely on single authorities, yet still achieve fine efficiency and privacy features.

**Efficiency.** The efficiency of our e-donation scheme is mainly affected by the underlying PrivCash and VABS schemes. Besides efficiency gains from relaxations of the framework requirements, improvements on both computation time and bandwidth use in both primitives will directly improve the efficiency of our scheme. Moreover,

<sup>11</sup> <https://nonprofitssource.com/online-giving-statistics/>

it may be possible to design a more efficient e-donation protocol that is independent of these building blocks.

## Acknowledgements

We acknowledge the support of TÜBİTAK (the Scientific and Technological Research Council of Turkey) project number 119E088. We also thank Markulf Kohlweiss for very insightful comments and helpful discussion.

## References

- [1] 2019. Enigma Web Site. <https://enigma.co/>. Accessed: 2020-03-05.
- [2] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek. 2016. Secure Multiparty Computations on Bitcoin. *Commun. ACM* 59, 4 (March 2016).
- [3] M. H. Au, W. Susilo, Y. Mu, and S. S. M. Chow. 2013. Constant-Size Dynamic K-Times Anonymous Authentication. *IEEE Systems Journal* 7, 2 (2013), 249–261.
- [4] L. Axon and M. Goldsmith. 2017. PB-PKI: A Privacy-aware Blockchain-based PKI. In *SECURITY ICETE '17*.
- [5] M. Bellare, H. Shi, and C. Zhang. 2005. Foundations of Group Signatures: The Case of Dynamic Groups. In *CT-RSA '05*.
- [6] I. Bentov and R. Kumaresan. 2014. How to Use Bitcoin to Design Fair Protocols. In *CRYPTO '14*.
- [7] O. Biçer and A. Küpçü. 2019. Versatile ABS: Usage Limited, Revocable, Threshold Traceable, Authority Hiding, Decentralized Attribute Based Signatures. Cryptology ePrint Archive, Report 2019/203. <https://eprint.iacr.org/2019/203>.
- [8] E.-O. Blass and F. Kerschbaum. 2017. Strain: A Secure Auction for Blockchains. Cryptology ePrint Archive, Report 2017/1044. <https://eprint.iacr.org/2017/1044>.
- [9] J. Bonneau, A. Narayanan, A. Miller, J. Clark, J. A. Kroll, and E. W. Felten. 2014. Mixcoin: Anonymity for bitcoin with accountable mixes. In *FC '14*.
- [10] S. Bove, A. Chiesa, M. Green, I. Miers, P. Mishra, and H. Wu. 2020. ZEXE: Enabling Decentralized Private Computation. In *IEEE SP*. IEEE Computer Society, Los Alamitos, CA, USA.
- [11] Beth Breeze. 2013. How Donors Choose Charities: The Role of Personal Taste and Experiences in Giving Decisions. *Voluntary Sector Review* 4 (07 2013), 165–183. <https://www.kent.ac.uk/sspsr/philanthropy/documents/How%20Donors%20Choose%20Charities%2018%20June%202010.pdf>.
- [12] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. 2019. Zether: Towards Privacy in a Smart Contract World. Cryptology ePrint Archive, Report 2019/191.
- [13] Vitalik Buterin. 2013. Ethereum White Paper: A next-generation smart contract and decentralized application platform. (2013).
- [14] J. Camenisch, M. Drijvers, and J. Hajny. 2016. Scalable Revocation Scheme for Anonymous Credentials Based on N-times Unlinkable Proofs. In *ACM WPES '16*.
- [15] J. Camenisch and T. Groß. 2012. Efficient Attributes for Anonymous Credentials. *ACM Trans. Inf. Syst. Secur.* 15, 1 (March 2012).
- [16] J. Camenisch and J. Groth. 2005. Group Signatures: Better Efficiency and New Theoretical Aspects. In *Security in Communication Networks*.
- [17] J. Camenisch, S. Hohenberger, M. Kohlweiss, A. Lysyanskaya, and M. Meyerovich. 2006. How to Win the Clonewars: Efficient Periodic N-times Anonymous Authentication. In *ACM CCS '06*.
- [18] J. Camenisch and A. Lysyanskaya. 2003. A Signature Scheme with Efficient Protocols. In *Security in Communication Networks*. Springer Berlin Heidelberg.
- [19] D. Cao, B. Zhao, X. Wang, and J. Su. 2012. Flexible Multi-authority Attribute-based Signature Schemes for Expressive Policy. *Mob. Inf. Syst.* 8, 3 (July 2012).
- [20] Cashlib 2010. <https://github.com/brownie/cashlib>. Accessed: 2020-02-15.
- [21] D. Chaum. 1983. Blind Signatures for Untraceable Payments. In *CRYPTO '82*.
- [22] Electric Coin Company. 2018. Zcash “Sapling” cryptography. <https://github.com/zcash-hackworks/sapling-crypto>. Accessed: 2020-02-28.
- [23] D. Derler, C. Hanser, and D. Slamanig. 2015. A New Approach to Efficient Revocable Attribute-Based Anonymous Credentials. In *IMACC '15*.
- [24] Constantin-Cătălin Drăgan, Daniel Gardham, and Mark Manulis. 2018. Hierarchical Attribute-Based Signatures. In *CNS '18*.
- [25] C. Dwork and M. Naor. 1993. Pricing via Processing or Combatting Junk Mail. In *CRYPTO '92*.
- [26] A. El Kaafarani, E. Ghadafi, and D. Khader. 2014. Decentralized Traceable Attribute-Based Signatures. In *CT-RSA '14*.
- [27] K. Emura, T. Hayashi, and A. Ishida. 2017. Group Signatures with Time-bound Keys Revisited: A New Model and an Efficient Construction. In *ASIA CCS '17*.
- [28] I. Eyal and E. G. Sirer. 2018. Majority is Not Enough: Bitcoin Mining is Vulnerable. *Commun. ACM* 61, 7 (June 2018).
- [29] J. A. Garay, A. Kiayias, and N. Leonardos. 2015. The Bitcoin Backbone Protocol: Analysis and Applications. In *EUROCRYPT '15*.
- [30] J. A. Garay, A. Kiayias, and N. Leonardos. 2017. The Bitcoin Backbone Protocol with Chains of Variable Difficulty. In *CRYPTO '17*.
- [31] C. Garman, M. Green, and I. Miers. 2014. Decentralized Anonymous Credentials. In *NDSS '14*.
- [32] B. Hampiholi, G. Alpár, F. v. d. Broek, and B. Jacobs. 2015. Towards Practical Attribute-Based Signatures. In *Security, Privacy, and Applied Cryptography Engineering*.
- [33] Yahya Hassanzadeh-Nazarabadi, Alptekin Küpçü, and Öznur Özkasap. 2019. LightChain: A DHT-based Blockchain for Resource Constrained Environments. *arXiv preprint arXiv:1904.00375* (2019). <http://arxiv.org/abs/1904.00375>

- [34] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. 2020. Zcash Protocol Specification. <https://raw.githubusercontent.com/zcash/zips/master/protocol/protocol.pdf>.
- [35] J. Hwang, L. Chen, H. Cho, and D. Nyang. 2015. Short Dynamic Group Signature Scheme Supporting Controllable Linkability. In *IEEE TIFS*, Vol. 10.
- [36] CAF (2018) World Giving Index. 2018. [https://www.cafonline.org/docs/default-source/about-us-publications/caf\\_wgi2018\\_report\\_webnopw\\_2379a\\_261018.pdf?sfvrsn=c28e9140\\_4](https://www.cafonline.org/docs/default-source/about-us-publications/caf_wgi2018_report_webnopw_2379a_261018.pdf?sfvrsn=c28e9140_4).
- [37] Harry Kalodner, Steven Goldfeder, Xiaoqi Chen, S. Matthew Weinberg, and Edward W. Felten. 2018. Arbitrum: Scalable, Private Smart Contracts. In *USENIX (SEC'18)*.
- [38] J. Katz and Y. Lindell. 2007. *Introduction to Modern Cryptography* (2nd ed.). Chapman & Hall/CRC.
- [39] A. Kiayias, E. Koutsoupias, M. Kyropoulou, and Y. Tseleounis. 2016. Blockchain Mining Games. In *ACM EC '16*.
- [40] A. Kiayias, H.-S. Zhou, and V. Zikas. 2016. Fair and Robust Multi-party Computation Using a Global Transaction Ledger. In *EUROCRYPT '16*.
- [41] Sunny King and Scott Nadal. 2012. PPScoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake.
- [42] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou. 2016. Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts. In *IEEE SP*.
- [43] M. Y. Kubilay, M. S. Kiraz, and H. A. Mantar. 2018. CertLedger: A New PKI Model with Certificate Transparency Based on Blockchain. *arXiv preprint arXiv:1806.03914* (2018). <http://arxiv.org/abs/1806.03914>.
- [44] R. Kumaresan and I. Bentov. 2014. How to Use Bitcoin to Incentivize Correct Computations. In *ACM CCS '14*.
- [45] R. Kumaresan, V. Vaikuntanathan, and P. N. Vasudevan. 2016. Improvements to Secure Computation with Penalties. In *ACM CCS '16*.
- [46] Leslie Lamport, Robert Shostak, and Marshall Pease. 1982. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.* 4, 3 (July 1982).
- [47] Litecoin 2011. <https://litecoin.org/>. Accessed: 2020-01-15.
- [48] W. Lueks, G. Alpár, J.-H. Hoepman, and P. Vullers. 2015. Fast Revocation of Attribute-Based Credentials for Both Users and Verifiers. In *ICT Systems Security and Privacy Protection*.
- [49] H. K. Maji, M. Prabhakaran, and M. Rosulek. 2011. Attribute-Based Signatures. In *CT-RSA '11*.
- [50] S. Meiklejohn, C. C. Erway, A. Küpçü, T. Hinkle, and A. Lysyanskaya. 2010. ZKPD: A Language-based System for Efficient Zero-knowledge Proofs and Electronic Cash (*USENIX Security '10*).
- [51] I. Miers, C. Garman, M. Green, and A. D. Rubin. 2013. Zerocoin: Anonymous Distributed E-Cash from Bitcoin. In *IEEE SP '13*. <http://zerocash-project.org/media/pdf/zerocash-extended-20140518.pdf>.
- [52] Monero 2014. <https://www.getmonero.org/>. Accessed: 2020-01-15.
- [53] M. Möser, K. Soska, E. Heilman, K. Lee, H. Heffan, S. Srivastava, K. Hogan, J. Hennessey, A. Miller, A. Narayanan, and N. Christin. 2018. An Empirical Analysis of Traceability in the Monero Blockchain. *PoPETs* (2018).
- [54] S. Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system, <http://bitcoin.org/bitcoin.pdf>.
- [55] M. Nisansala, S. Perera, and T. Koshiha. 2017. Fully secure lattice-based group signatures with verifier-local revocation. In *IEEE, AINA '17*.
- [56] T. Okamoto and K. Takashima. 2013. Decentralized Attribute-Based Signatures. In *PKC '13*.
- [57] S. Popov. 2017. The Tangle. [http://www.iota.org/IOTA\\_Whitepaper.pdf](http://www.iota.org/IOTA_Whitepaper.pdf).
- [58] N. Roby. 2017. Application of Blockchain technology in online voting. [https://www.rsaconference.com/writable/files/About/application\\_of\\_blockchain\\_technology\\_in\\_online\\_voting.pdf](https://www.rsaconference.com/writable/files/About/application_of_blockchain_technology_in_online_voting.pdf).
- [59] T. Ruffing, P. Moreno-Sanchez, and A. Kate. 2014. Coin-Shuffle: Practical Decentralized Coin Mixing for Bitcoin. In *ESORICS '14*.
- [60] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. 2014. Zerocash: Decentralized Anonymous Payments from Bitcoin. In *IEEE SP '14*.
- [61] F. R. Schreier. 1973. *Sybil*. Warner Books.
- [62] D. Slamanig, R. Spreitzer, and T. Unterluggauer. 2014. Adding Controllable Linkability to Pairing-Based Group Signatures for Free. In *Information Security*.
- [63] Yonatan Sompolsky, Yoad Lewenberg, and Aviv Zohar. 2017. SPECTRE : Serialization of Proof-of-work Events : Confirming Transactions via Recursive Elections.
- [64] Yonatan Sompolsky and Aviv Zohar. 2018. PHANTOM , GHOSTDAG : Two Scalable BlockDAG protocols.
- [65] Paul Syverson, R. Dingledine, and N. Mathewson. 2004. Tor: The second generation onion router. In *Usenix Security*.
- [66] Florian Tramèr, Dan Boneh, and Kenneth G. Paterson. 2020. Remote Side-Channel Attacks on Anonymous Transactions. *Cryptology ePrint Archive*, Report 2020/220.
- [67] M. Urquidi, D. Khader, J. Lancrenon, and L. Chen. 2016. Attribute-Based Signatures with Controllable Linkability. In *Trusted Systems*.
- [68] N. v. Saberhagen. 2013. CryptoNote v 2.0, <https://cryptonote.org/whitepaper.pdf>.
- [69] D. Yaga, P. Mell, N. Roby, and K. Scarfone. 2018. *Blockchain Technology Overview*. Technical Report. NIST. <https://nvlpubs.nist.gov/nistpubs/ir/2018/NIST.IR.8202.pdf>.
- [70] A. Yakubov, W. M. Shbair, A. Wallbom, D. Sanda, and R. State. 2018. A Blockchain-Based PKI Management Framework. *IEEE/IFIP NOMS '18*.
- [71] G. Yang, S. Tang, and L. Yang. 2011. A Novel Group Signature Scheme Based on MPKC. In *ISPEC '11*.
- [72] J. H. Ziegeldorf, F. Grossmann, M. Henze, N. Inden, and K. Wehrle. 2015. CoinParty: Secure Multi-Party Mixing of Bitcoins. In *ACM CODASPY '15*.

## A Security Proof of Our Scheme

We now show that our e-donation scheme satisfies the security requirements of our e-donation framework, and prove Theorem 1 by providing partial proofs for each of unforgeability and balance, fair distribution of do-

nations, and donation and recipient privacy definitions given in the generic e-donation framework in Section 3 via formal reductions to the security assumptions on the underlying primitives.

## A.1 Proof of Unforgeability and Balance

**Lemma 1.** *If the underlying PrivCash satisfies the balance and non-malleability property (see Appendix B), the digital signature scheme utilized for signing PrivCashMint is existentially unforgeable under an adaptive chosen-message attack [38], and the underlying VABS protocol satisfies signature unforgeability defined in [7] (see Appendix D); then our e-donation scheme is unforgeable and balanced.*

*Proof.* Applying the definition of unforgeability and balance to our scheme, the adversary can win by generating one of the following transactions:

- a  $\text{TX}_{\text{Donate},1}$  whose references include at least one of the transactions  $(v_1, p_1), \dots, (v_\ell, p_\ell)$  of honest users (corresponding to the winning condition (1)),
- a  $\text{TX}_{\text{Donate},2}$  that spends a PrivCash other than an unspent one of the donor of the related  $\text{TX}_{\text{Donate},1}$ , or that can replace an existing transaction  $\text{TX}'_{\text{Donate},2}$  with a different policy on the Ledger generated by a user under  $\mathcal{C}$ 's control (and the verification algorithm verifies all the transactions on the Ledger upto and including  $\text{TX}_{\text{Donate},2}$ ) (corresponding to the winning condition (2), since the amount of the minted PrivCash by  $\text{TX}_{\text{Donate},1}$  cannot exceed the total amount of its referenced BaseCash due to the scanning check by the Ledger writer and the output only adds to the donations with the related  $\beta$ ),
- a  $\text{TX}_{\text{Claim},1}$  that references a  $\text{TX}_{\text{Donate},2}$  with policy  $\beta \wedge \omega$  (corresponding to the winning condition (3)),
- a  $\text{TX}_{\text{Claim},2}$  that spends a PrivCash other than an unspent one of the recipient of the related  $\text{TX}_{\text{Claim},1}$ , or that can replace an existing transaction tuple  $\text{TX}'_{\text{Claim},2}$  on the Ledger generated by a user under  $\mathcal{C}$ 's control such that  $\text{TX}'_{\text{Claim},2} \neq \text{TX}_{\text{PrivateDonate}}$  (and the verification algorithm verifies all the transactions on the Ledger upto and including  $\text{TX}_{\text{Donate},k}$ ) (corresponding to the winning condition (4), since the amount of the minted PrivCash by  $\text{TX}_{\text{Claim},1}$  cannot exceed the total amount of its referenced  $\text{TX}_{\text{Donat}2,2}$  due to the scanning check by the Ledger writer and the output only adds to the balance of the payee  $p$ ).

If a PPT adversary  $\hat{\mathcal{A}}$  wins the game eDonForge with non-negligible probability, we can utilize it to construct a PPT algorithm  $\hat{\mathcal{B}}$  who wins the signature existential unforgeability game (Sig-forge in [38]) or the BalanceTNM game or the VABSForge game with non-negligible advantage. In eDonForge,  $\hat{\mathcal{A}}$  plays the role of  $\mathcal{A}$ , and  $\hat{\mathcal{B}}$  plays the role of  $\mathcal{C}$ . In Sig-forge, BalanceTNM, and VABSForge games,  $\hat{\mathcal{B}}$  plays the role of  $\mathcal{A}$ . We denote the honest challengers of them as  $\hat{\mathcal{C}}_f$ ,  $\hat{\mathcal{C}}_b$ , and  $\hat{\mathcal{C}}_a$ , respectively.

1. In Sig-forge, using the security parameter  $1^\lambda$ ,  $\hat{\mathcal{C}}_f$  generates a key pair  $(p, s)$ . It then gives  $1^\lambda, p$ , and access to signing oracle with  $s$  to  $\hat{\mathcal{B}}$ .
2. In BalanceTNM,  $\hat{\mathcal{B}}$  receives  $1^\lambda$  (assumed to be the same as in Step 1 above), the unit donation amount  $u$ ,  $pparams$ , and query access to  $\mathcal{O}$ , from  $\hat{\mathcal{C}}_b$ .
3. In VABSForge,  $\hat{\mathcal{B}}$  receives  $1^\lambda$  (assumed to be the same as in Step 1 above),  $n, \delta, vparams$  (Step 1 of VABSForge).
4. In eDonForge,  $\hat{\mathcal{B}}$  generates  $(\hat{s}, \hat{p})$  gives to  $\hat{\mathcal{A}}$  the values  $1^\lambda, dparams := (vparams, pparams, \hat{s}, \hat{p})$ , and read access to all Ledger content.  $\hat{\mathcal{B}}$  gives  $u, D := n \cdot u$ , and  $\delta$  to  $\hat{\mathcal{A}}$  (Step 1 of eDonForge).
5. In eDonForge,  $\hat{\mathcal{B}}$  follows Step 2 of the game with  $\hat{\mathcal{A}}$  in the exact same way as an honest  $\mathcal{C}$  would do, except for the following: Whenever  $\hat{\mathcal{A}}$  requests from  $\hat{\mathcal{B}}$  an identity provider or attribute authority oracle generation, in VABSForge,  $\hat{\mathcal{B}}$  requests an authority from  $\hat{\mathcal{C}}_a$ .  $\hat{\mathcal{B}}$  obtains the public key of the authority from  $\hat{\mathcal{C}}_a$ , and gives it to  $\hat{\mathcal{A}}$  as response. Further, if  $\hat{\mathcal{A}}$  request him to generate users,  $\hat{\mathcal{A}}$  queries  $\hat{\mathcal{C}}_b$  with PrivCashKeyGen, and forwards the output to  $\hat{\mathcal{A}}$ . Whenever  $\hat{\mathcal{A}}$  queries AttrGen or RecipJoin oracle,  $\hat{\mathcal{B}}$  queries the related authority in VABSForge (Step 2 of VABSForge).  $\hat{\mathcal{B}}$  obtains the output, and returns it back to  $\hat{\mathcal{A}}$ . We note that  $\hat{\mathcal{B}}$  keeps a table that matches each user public key generated by PrivCashKeyGen query in BalanceTNM and the secret key generated by  $\hat{\mathcal{B}}$  for RecipJoin. For generation of BaseCash by the users under  $\hat{\mathcal{B}}$ 's control,  $\hat{\mathcal{B}}$  generates them itself, and queries  $\mathcal{O}$  with Insert of BaseCash. For generation of  $\text{TX}_{\text{Donate},1}$ ,  $\text{TX}_{\text{Donate},2}$ , and  $\text{TX}_{\text{Claim},2}$  by the users under  $\hat{\mathcal{B}}$ 's control,  $\hat{\mathcal{B}}$  straightforwardly queries  $\mathcal{O}$  with types PrivCashMint, PrivCashSpend (with  $\hat{p}$ ), and PrivCashSpend, respectively. It then writes the transactions generated on the Ledger of BalanceTNM to the Ledger of eDonForge. Regarding generation of  $\text{TX}_{\text{Claim},1}$ , first  $\hat{\mathcal{B}}$  queries  $\mathcal{O}$  with PrivCashMint (with  $\dots s$ ), and obtains the resulting transaction

as  $\text{TX}_{\text{PrivCashMint}}$ .  $\hat{\mathcal{B}}$  then generates VABS  $\sigma'$  on  $\text{TX}_{\text{PrivCashMint}}$ , and writes the resulting transaction  $(\text{TX}_{\text{PrivCashMint}}, \sigma')$  to the Ledger of eDonForge. The transactions BaseCash,  $\text{TX}_{\text{Donate},1}$ ,  $\text{TX}_{\text{Donate},2}$ , and  $\text{TX}_{\text{Claim},2}$  generated by  $\hat{\mathcal{A}}$  are inserted (i.e., via Insert queries) to the Ledger of BalanceTNM as BaseCash PrivCashMint, PrivCashSpend, and PrivCashSpend, respectively. Regarding Insert with  $\text{TX}_{\text{Claim},1}$ , first  $\hat{\mathcal{B}}$  drops VABS, and then queries  $\mathcal{O}$  for Insert of the resulting transaction (Step 1 of BalanceTNM).

6. In eDonForge,  $\hat{\mathcal{B}}$  generates some BaseCash tuples  $(v_1, p), (v_2, p_2), \dots, (v_\ell, p_\ell)$  with  $\hat{\mathcal{A}}$ 's request, and writes them to the Ledger, while keeping the secret keys private (Step 3 of eDonForge).
7. In eDonForge,  $\hat{\mathcal{A}}$  returns  $\omega$  to  $\hat{\mathcal{B}}$  (Step 4 of eDonForge).
8. In VABSSound,  $\hat{\mathcal{B}}$  gives  $\omega$  to  $\hat{\mathcal{C}}_a$  (Step 3 of VABSSound).
9. Step 4 of VABSSound is conducted by  $\hat{\mathcal{B}}$  by forwarding messages between  $\hat{\mathcal{C}}_a$  and  $\hat{\mathcal{A}}$ .
10. In eDonForge,  $\hat{\mathcal{A}}$  eventually outputs a transaction  $\text{TX}_{\text{Donate},1}$  whose references include a subset of  $\{(v_1, p), (v_2, p_2), \dots, (v_\ell, p_\ell)\}$ , or a transaction  $\text{TX}_{\text{Donate},2}$ , or a transaction  $\text{TX}_{\text{Claim},1}$  that references a  $\text{TX}_{\text{Donate},2}$  with policy  $\beta \wedge \omega$ , or a transaction  $\text{TX}_{\text{Claim},2}$ .  $\hat{\mathcal{A}}$  sends this transaction to  $\hat{\mathcal{B}}$  (Step 5 of eDonForge).
  - (a) If this transaction is a  $\text{TX}_{\text{Donate},1}$ , assuming that it particularly references  $(v_1, p)$ , in Sig-forged,  $\hat{\mathcal{B}}$  parses  $\text{TX}_{\text{Donate},1}$  for the digital signature  $\sigma$  and the signed transcript  $(\text{TX}_{\text{Donate},1}')$ , possible as  $\text{TX}_{\text{Donate},1}$  is obtained via a plain PrivCashMint operation.
  - (b) If this transaction is a  $\text{TX}_{\text{Donate},2}$  or  $\text{TX}_{\text{Claim},2}$ , it is output as the transaction  $\text{TX}'_{\text{PrivCashSpend}}$  in BalanceTNM (Step 2 of BalanceTNM).
  - (c) If this transaction is a  $\text{TX}_{\text{Claim},1}$ , in VABSSound,  $\hat{\mathcal{B}}$  parses  $(\text{TX}_{\text{Claim},1}, \sigma') := \text{TX}_{\text{Claim},1}$  and outputs  $(\text{TX}_{\text{Claim},1}, \beta \wedge \omega, \sigma')$  (Step 5 of VABSSound).

Regarding the case (a), the output of the game Sig-forged is exactly the same as that of eDonForge. Namely,  $\hat{\mathcal{B}}$  wins Sig-forged with non-negligible advantage, if  $\hat{\mathcal{A}}$  wins eDonForge with non-negligible advantage.

Regarding the case (b), if  $\hat{\mathcal{A}}$  can win eDonForge with non-negligible probability, then it succeeds in casting an imbalanced  $\text{TX}_{\text{Donate},2}$  or  $\text{TX}_{\text{Claim},2}$  whose amount exceeds the balance of BaseCash tuples referenced, or in forging an existing one of its type. The output of the game BalanceTNM is exactly the same as that of eDonForge. Namely,  $\hat{\mathcal{B}}$  wins BalanceTNM with non-

negligible advantage, if  $\hat{\mathcal{A}}$  wins eDonForge with non-negligible advantage.

Regarding the case (c), due to direct referencing and check by the Ledger writer,  $\hat{\mathcal{A}}$  cannot generate an invalid Claim whose amount exceeds  $u$ . The only way for  $\hat{\mathcal{A}}$  to win eDonForge with non-negligible probability is claiming a  $\text{TX}_{\text{Donate},2}$  without having the policy requirement. The output of the game VABSSound is exactly the same as that of eDonForge. Namely,  $\hat{\mathcal{B}}$  wins VABSSound with non-negligible advantage, if  $\hat{\mathcal{A}}$  wins eDonForge with non-negligible advantage.

All in all, if  $\hat{\mathcal{A}}$  wins eDonForge with non-negligible advantage, then  $\hat{\mathcal{B}}$  wins Sig-forged, or BalanceTNM, or VABSSound with non-negligible advantage, concluding the proof.  $\square$

## A.2 Proof of Fair Distribution of Donations

**Lemma 2.** *If the underlying VABS protocol satisfies signature soundness defined in [7] (see Appendix D), then our e-donation scheme provides fair distribution of donations.*

*Proof.* We now reduce the fair distribution of donations in our e-donation scheme to the signature soundness of the underlying VABS. If a PPT adversary  $\hat{\mathcal{A}}$  wins the game FairDist with non-negligible advantage, then we can utilize  $\hat{\mathcal{A}}$  to construct a PPT algorithm  $\hat{\mathcal{B}}$  who wins the game VABSSound with non-negligible advantage. In the FairDist game,  $\hat{\mathcal{A}}$  and  $\hat{\mathcal{B}}$  play the roles of  $\mathcal{A}$  and  $\mathcal{C}$ , respectively.  $\hat{\mathcal{B}}$  also plays the role of  $\mathcal{A}$  against the honest challenger  $\hat{\mathcal{C}}$  in VABSSound.

1. In VABSSound,  $\hat{\mathcal{B}}$  receives  $1^\lambda$ ,  $n$ ,  $\delta$ , and  $vparams$  from  $\hat{\mathcal{C}}$  (Step 1 of VABSSound).
2. In FairDist,  $\hat{\mathcal{B}}$  itself picks an arbitrary  $u \in poly(\lambda)$  and generates  $pparams$  and  $(\hat{s}, \hat{p})$  using  $1^\lambda$ .  $\hat{\mathcal{B}}$  gives to  $\hat{\mathcal{A}}$  the values  $1^\lambda$ ,  $D := n \cdot u$ ,  $\delta$ ,  $dparams := (vparams, pparams, \hat{s}, \hat{p})$ , and read access to all Ledger content (Step 1 of FairDist).
3. In FairDist,  $\hat{\mathcal{B}}$  follows Step 2 of the game with  $\hat{\mathcal{A}}$  in the exact same way as an honest  $\mathcal{C}$  would do, except for the following: Whenever  $\hat{\mathcal{A}}$  requests from  $\hat{\mathcal{B}}$  an attribute authority oracle generation, in VABSSound,  $\hat{\mathcal{B}}$  requests an attribute authority from  $\hat{\mathcal{C}}$ .  $\hat{\mathcal{B}}$  obtains the public key of the authority from  $\hat{\mathcal{C}}$ , and gives it to  $\hat{\mathcal{A}}$  as response. Similarly, whenever  $\hat{\mathcal{A}}$  queries AttrGen oracle,  $\hat{\mathcal{B}}$  queries the related author-

ity in VABSSound (Step 2 of VABSSound).  $\hat{B}$  obtains the output, and returns it back to  $\hat{A}$ .

4. In FairDist, if  $\hat{A}$  wins (i.e.,  $\hat{A}$  comes up with a transaction tuple for PrivateClaim $_{\hat{A}}$  whose recipient with secret key  $s$  has claimed at least  $D$  amount in the current time period, and the transaction tuple is verified for being written to the Ledger),  $\hat{B}$  parses each TX $_{\text{Claim},1} = (r, c, \sigma)$  transaction that is written to the Ledger as  $(r, c)$  and VABS  $\sigma$  (Step 3 of FairDist).
5. In VABSSound,  $\hat{B}$  gives the message  $(r, c)$  and VABS  $\sigma$  pairs to  $\hat{C}$  for verification (Step 3 of VABSSound).

If  $\hat{A}$  can win FairDist with non-negligible probability, then it succeeds in minting (via TX $_{\text{Claim},1}$  transactions) more than  $n \cdot u$  amount PrivCash for a specific user (i.e., signing more than  $n$  messages with the same  $s$  that gets verified). The output of the game VABSSound is exactly the same as that of FairDist, since Verify algorithm run on the message and ABS pairs of those TX $_{\text{Claim},1}$ s outputs 1. Namely,  $\hat{B}$  wins VABSSound with non-negligible advantage if  $\hat{A}$  wins FairDist with non-negligible advantage.  $\square$

### A.3 Proof of Donation and Recipient Privacy

**Lemma 3.** *If the underlying PrivCash satisfies the ledger indistinguishability of [60], and the underlying VABS protocol satisfies anonymity defined in [7] (see Appendix D); then our e-donation scheme provides donation and recipient identity privacy.*

*Proof.* We utilize the following deductions from the Lemma 3. Its contrapositive:

If our e-donation scheme does not provide donation and recipient privacy; then the underlying PrivCash does not satisfy the ledger indistinguishability of [60], or the underlying VABS protocol does not satisfy anonymity defined in [7] (see Appendix D).

From the above form of Lemma 3, we can obtain:

If our e-donation scheme does not provide donation and recipient privacy, and the underlying VABS protocol satisfies anonymity defined in [7] (see Appendix D); then the underlying PrivCash does not satisfy the ledger indistinguishability of [60].

Assuming that the underlying VABS satisfies anonymity of [7], we now reduce the donation and recipient privacy of our e-donation scheme to the ledger indis-

tinguishability of the underlying PrivCash. If a PPT adversary  $\hat{A}$  wins the donation and recipient privacy game Privacy with non-negligible advantage, then we can utilize  $\hat{A}$  to construct a PPT algorithm  $\hat{B}$  that wins ledger indistinguishability game L-IND with non-negligible advantage. In the game Privacy,  $\hat{A}$  and  $\hat{B}$  play the roles of  $\mathcal{A}$  and  $\mathcal{C}$ , respectively. In the L-IND game,  $\hat{B}$  plays the role of  $\mathcal{A}$  against an honest challenger  $\hat{C}$ .

1. In L-IND, Steps 1, 2, and 3 of the game are followed between  $\hat{B}$  and  $\hat{C}$ .
2. In Privacy,  $\hat{B}$  gives to  $\hat{A}$   $1^\lambda$  that is obtained in L-IND.  $\hat{B}$  runs the setup operation of VABS to generate  $vparams$  and generates  $(\hat{s}, \hat{p})$ . Then,  $\hat{B}$  sets  $dparams = (vparams, pparams, \hat{s}, \hat{p})$  and gives  $dparams$  to  $\hat{A}$ .  $\mathcal{A}$  is also given the unit donation amount  $u$  obtained in L-IND, the total donation amount  $D = n \cdot u$  that can be received by a recipient in a given time period where  $n \in poly(\lambda)$ , and the duration  $\delta$  of a time period. Time period counter  $t$  is initialized as  $t := 1$ , and is started (Step 1 of Privacy).
3. In Privacy,  $\hat{B}$  generates all the authorities and runs their oracles itself. Regarding UserJoin for the users under  $\hat{B}$ 's control  $\hat{B}$  queries both  $Q_0$  and  $Q_1$  as PrivCashKeyGen in L-IND. Regarding RecipJoin for the users under  $\hat{B}$ 's control,  $\hat{B}$  generates the recipient secret key  $S$  itself.  $\hat{B}$  keeps a table that matches each user public key generated by PrivCashKeyGen query in L-IND and the secret key generated by  $\hat{B}$  for RecipJoin (Step 2 of Privacy).
4. In Privacy,  $\hat{B}$  sets Ledger $_{\text{left}}$  and Ledger $_{\text{right}}$  essentially as Ledger $_{\text{left}}$  and Ledger $_{\text{right}}$  of L-IND (Step 3 of Privacy).
5.  $\hat{B}$  forward Donate, 1, Donate, 2, and Claim, 2 queries in Privacy as PrivCashMint, PrivCashSpend (with  $\hat{p}$ ), and PrivCashSpend queries to  $\hat{C}$  in L-IND without changing the order of  $(Q_0, Q_1)$ , respectively.  $\hat{B}$  then writes to Ledger $_{\text{left}}$  and Ledger $_{\text{right}}$  in Privacy the resulting transaction on Ledger $_{\text{left}}$  and Ledger $_{\text{right}}$  in L-IND, respectively. Regarding BaseCash queries for the users under  $\hat{B}$ 's control,  $\hat{B}$  generates them itself, and queries in L-IND with Insert of BaseCash without changing the order of  $(Q_0, Q_1)$ . Regarding Claim, 1 queries, first  $\hat{B}$  queries them in L-IND as PrivCashMint with  $\hat{s}$  without changing the order of  $(Q_0, Q_1)$ .  $\hat{B}$  obtains the resulting transactions TX $_{\text{PrivCashMint},\text{left}}$  and TX $_{\text{PrivCashMint},\text{right}}$  in L-IND.  $\hat{B}$  then generates VABSs  $\sigma'_{\text{left}}$  and  $\sigma'_{\text{right}}$  on TX $_{\text{PrivCashMint},\text{left}}$  and TX $_{\text{PrivCashMint},\text{right}}$ , and writes the resulting

transactions  $\text{TX}_{\text{Claim, left}} := (\text{TX}_{\text{PrivCashMint, left}}, \sigma'_{\text{left}})$  and  $\text{TX}_{\text{Claim, right}} := (\text{TX}_{\text{PrivCashMint, right}}, \sigma'_{\text{right}})$  to the  $\text{Ledger}_{\text{left}}$  and  $\text{Ledger}_{\text{right}}$  of Privacy. The transactions  $\text{BaseCash}$ ,  $\text{TX}_{\text{Donate, 1}}$ ,  $\text{TX}_{\text{Donate, 2}}$ , and  $\text{TX}_{\text{Claim, 2}}$  inserted (i.e., via  $\text{Insert}$  queries) by  $\hat{\mathcal{A}}$  to the Ledger of Privacy are inserted to the Ledger of L-IND as  $\text{BaseCash}$ ,  $\text{PrivCashMint}$ ,  $\text{PrivCashSpend}$ , and  $\text{PrivCashSpend}$ , respectively. Regarding  $\text{Insert}$  with  $\text{TX}_{\text{Claim, 1s}}$ , first  $\hat{\mathcal{B}}$  drops VABSs, and then queries for  $\text{Insert}$  of the resulting transactions in L-IND without changing the order of  $(Q_0, Q_1)$  (Step 4 of Privacy).

6.  $\hat{\mathcal{B}}$  eventually outputs in Privacy the bit that  $\hat{\mathcal{A}}$  eventually outputs in Privacy.

In the above scenario,  $\hat{\mathcal{B}}$  does not directly pick a bit, but implicitly sets it as the bit that is picked by  $\hat{\mathcal{C}}$ . We highlight that  $\text{Ledger}_{\text{left}}$  and  $\text{Ledger}_{\text{right}}$  of Privacy and those of L-IND given here are only different in terms of VABSs of the transactions  $\text{TX}_{\text{Claim, 2}}$ . As the underlying VABS scheme is assumed to satisfy anonymity of [7], if  $\hat{\mathcal{A}}$  wins the game Privacy, then  $\hat{\mathcal{B}}$  wins the game L-IND. If  $\hat{\mathcal{A}}$  wins the game Privacy with  $1/2$  plus non-negligible probability, then  $\hat{\mathcal{B}}$  wins the game L-IND with  $1/2$  plus non-negligible probability. We note that in case the underlying Privacy scheme is assumed to satisfy L-IND, it is trivial to utilize an adversary obtaining non-negligible advantage in Privacy in construction of one obtaining that in VABSAnonym.  $\square$

We note that our security definitions and construction do not model timing attacks (e.g., the ones in [66] for de-anonymization), as in ZeroCoin [51] and Zerocash [60]. Although a full solution to such side channel attacks is out of scope of this paper, one needs to ensure to have resolved them before deployment of e-donation.

## B Balance and Non-Malleability of PrivCash

The definition we provide here combines the “balance” definition of [51] (we also make this adaptive) and the “transaction non-malleability” definition of [60]. Also it is simplified, as we require a fixed unit amount  $u$  for each transaction and by omitting  $\text{Receive}$  operation (for generality, as  $\text{Receive}$  does not result in a transaction on the Ledger) and nested  $\text{PrivCash}$  spending (i.e., the output of  $\text{Pour}$  cannot be some  $\text{PrivCash}$  but some  $\text{BaseCash}$ ). Given a PPT adversary  $\mathcal{A}$ , a challenger  $\mathcal{C}$ , a  $\text{PrivCash}$

scheme, and a security parameter  $\lambda$ , consider the following  $\text{BalanceTNM}$  game:

1.  $\mathcal{A}$  is given  $1^\lambda$ , the unit donation amount  $u$ , read access to the Ledger, and the global setup parameters  $pparams$ .  $\mathcal{A}$  is also given query access to a Ledger writer oracle  $\mathcal{O}$  for which the allowed query types by  $\mathcal{A}$  and the oracle’s consequent actions are as follows:
  - $\text{PrivCashKeyGen}$ :  $\mathcal{O}$  generates a user with private/public key pair  $s, p$  by running the algorithm  $\text{PrivCashKeyGen}$ . It gives  $p$  to  $\mathcal{A}$  and keeps  $s$  private.
  - $(\text{PrivCashMint}, r, s, p)$ :  $r$  is the referenced  $\text{BaseCash}$ ,  $s$  is the related private key to the referenced  $\text{BaseCash}$ , and  $p$  is the public key of a user under  $\mathcal{C}$ ’s control generated by  $\text{PrivCashKeyGen}$ . If  $r$  is unspent,  $\mathcal{O}$  generates a  $\text{TX}_{\text{PrivCashMint}}$  transaction referencing  $r$ .
  - $(\text{PrivCashSpend}, r, p)$ :  $r$  is a reference to the related  $\text{TX}_{\text{PrivCashMint}}$  and  $p$  is the public key of the recipient. If no previous transaction has referenced  $r$ ,  $\mathcal{O}$  generates a  $\text{TX}_{\text{PrivCashSpend}}$  transaction referencing  $r$  spent to the recipient  $p$ .
  - $(\text{Insert}, \text{TX}/\text{BaseCash})$  where  $\text{TX}$  is any type of transaction among  $\text{TX}_{\text{PrivCashMint}}$  and  $\text{TX}_{\text{PrivCashSpend}}$ . If the query input includes  $\text{BaseCash}$  or the verification algorithm verifies the transaction,  $\mathcal{O}$  accepts.

$\mathcal{O}$  generates transactions as a result of  $\mathcal{A}$ ’s  $\text{BaseCash}$ ,  $\text{PrivCashMint}$ , and  $\text{PrivCashSpend}$  type of queries and writes them to the Ledger. If the type of a query is  $\text{Insert}$  and the result of  $\mathcal{O}$ ’s action is accept, the related  $\text{BaseCash}$  or transaction is written to the Ledger.

2. Eventually,  $\mathcal{A}$  returns to  $\mathcal{C}$  a transaction  $\text{TX}'_{\text{PrivCashSpend}}$ .  $\mathcal{A}$  wins (i.e., the game output is defined as 1), if one of the following two conditions holds: (1) the verification algorithm verifies  $\text{TX}'_{\text{PrivCashSpend}}$  with the current  $\text{Ledger}_{\mathcal{C}}$  and the union of  $\text{TX}'_{\text{PrivCashSpend}}$  and inserted  $\text{TX}_{\text{PrivCashSpend}}$  transactions (via  $\text{Insert}$  queries) on the Ledger exceeds the total amount of inserted  $\text{TX}_{\text{PrivCashMint}}$  transactions (via  $\text{Insert}$  queries) on the Ledger, or (2)  $\text{TX}'_{\text{PrivCashSpend}}$  spends a previously spent  $\text{PrivCash}$  of a user under  $\mathcal{C}$ ’s control by a transaction  $\text{TX}''_{\text{PrivCashSpend}}$  such that  $\text{TX}''_{\text{PrivCashSpend}} \neq \text{TX}'_{\text{PrivCashSpend}}$ , and the verification algorithm verifies  $\text{TX}'_{\text{PrivCashSpend}}$  with the portion of Ledger preceding  $\text{TX}'_{\text{PrivCashSpend}}$ .

**Definition 4** (Balance and Non-Malleability).  $\mathcal{A}$   $\text{PrivCash}$  scheme satisfies the balance and non-

*malleability property, for all PPT adversaries  $\mathcal{A}$ , there exists a negligible function  $n(\cdot)$  such that*

$$\Pr[\text{BalanceTNM} = 1] < n(\lambda)$$

For the sake of simplicity, our e-donation proof utilize the definition given here.

## C Ledger Indistinguishability of PrivCash

Regarding ledger indistinguishability, we use the game of [60]. Yet, we have simplified the definition of [60], since some features (e.g., Receive, and nested Pour operations, and referencing multiple coins in spending) come by Zerocash are not generalizable to other decentralized anonymous e-cash schemes, and are not utilized in our definitions. Yet, the complete Zerocash scheme satisfies our core security proofs as well. Given a PrivCash scheme, a challenger  $\mathcal{C}$ , a PPT adversary  $\mathcal{A}$ , and a security parameter  $\lambda$ , consider the following L-IND game:

1.  $\mathcal{C}$  picks a bit  $b \leftarrow \{0, 1\}$ . It also initializes two ledger writer oracles  $\mathcal{O}_0$  and  $\mathcal{O}_1$  which writes to ledgers  $\text{Ledger}_0$  and  $\text{Ledger}_1$ , respectively.
2.  $\mathcal{A}$  is given  $1^\lambda$ , the global setup parameters  $pparams$ , and the unit PrivCash amount  $u \in poly(\lambda)$ .  $\mathcal{A}$  is also given read access to the ledgers  $\text{Ledger}_{\text{left}}$  and  $\text{Ledger}_{\text{right}}$ , where  $\text{Ledger}_{\text{left}} := \text{Ledger}_b$  and  $\text{Ledger}_{\text{right}} := \text{Ledger}_{\bar{b}}$ .
3.  $\mathcal{A}$  sends its queries to  $\mathcal{C}$ . Each query  $Q$  must be in the form of a pair of subqueries  $(Q_0, Q_1)$ .  $\mathcal{C}$  forwards one of them to  $\mathcal{O}_0$  and the other one to  $\mathcal{O}_1$ . Allowed query types by  $\mathcal{A}$  and the forwarded oracle  $\mathcal{O}$ 's consequent actions are as follows:
  - PrivCashKeyGen:  $\mathcal{O}$  generates a user with private/public key pair  $s, p$  by running the algorithm PrivCashKeyGen. It gives  $p$  to  $\mathcal{A}$  and keeps  $s$  private.
  - (PrivCashMint,  $r, s, p$ ):  $r$  is the referenced BaseCash,  $s$  is the related private key to the referenced BaseCash, and  $p$  is the public key of a user under  $\mathcal{C}$ 's control generated by PrivCashKeyGen. If  $r$  is unspent,  $\mathcal{O}$  generates a  $\text{TX}_{\text{PrivCashMint}}$  transaction referencing  $r$ .
  - (PrivCashSpend,  $r, p$ ):  $r$  is a reference to the related  $\text{TX}_{\text{PrivCashMint}}$  and  $p$  is the public key of the recipient. If no previous transaction has referenced  $r$ ,  $\mathcal{O}$  generates a  $\text{TX}_{\text{PrivCashSpend}}$  transaction referencing  $r$  spent to the recipient  $p$ .

- (Insert, TX/BaseCash) where TX is any type of transaction among  $\text{TX}_{\text{PrivCashMint}}$  and  $\text{TX}_{\text{PrivCashSpend}}$ . If the query input includes BaseCash or the verification algorithm verifies the transaction,  $\mathcal{O}$  accepts.

Insert type of queries are for directly writing to the ledgers. Therefore, in this case, the subqueries  $Q_0$  and  $Q_1$  are forwarded to  $\mathcal{O}_b$  and  $\mathcal{O}_{\bar{b}}$  by  $\mathcal{C}$ , respectively. Other queries are for generation of the corresponding transactions. Therefore, in this case,  $\mathcal{C}$  forwards  $Q_0$  to  $\mathcal{O}_0$  and  $Q_1$  to  $\mathcal{O}_1$ . We require the following *public consistency* restrictions of [60]) for each query  $Q = (Q_0, Q_1)$  to deter trivial de-anonymization.

- $Q_0$  and  $Q_1$  must be of the same type.
  - If both oracles generate transactions as a result, each oracle writes the generated transaction to the corresponding ledger. Otherwise, both transactions are dropped.
  - If both oracles accept as a result of Insert, each oracle writes the accepted transaction or BaseCash to the corresponding ledger. Otherwise, both transactions are dropped.
  - If the type is PrivCashMint, the reference  $r$  and the public key  $p$  must be the same in  $Q_0$  and  $Q_1$ .
  - If the type is PrivCashSpend, the the recipient  $p$  and the public information info must be the same in  $Q_0$  and  $Q_1$ .
4. Eventually,  $\mathcal{A}$  outputs a bit  $b'$ .  $\mathcal{A}$  wins (i.e., the game output is defined as 1), if  $b = b'$ . Otherwise, the  $\mathcal{A}$  loses (i.e., the game output is defined as 0).

**Definition 5** (Ledger Indistinguishability). *A PrivCash scheme satisfies the ledger indistinguishability property, if for all PPT adversaries  $\mathcal{A}$ , there exists a negligible function  $n(\cdot)$  such that*

$$\Pr[\text{L-IND} = 1] < \frac{1}{2} + n(\lambda)$$

For the sake of simplicity, our e-donation proof utilize the definition given here.

## D Security Definitions of VABS

We provide the anonymity, signature unforgeability, and signature soundness definitions of [7] for completeness.

Given a VABS scheme  $\Pi$ , a PPT adversary  $\mathcal{A}$ , a challenger  $\mathcal{C}$ , a unary security parameter  $1^\lambda$ , and a limit  $n$  for a time period duration  $\delta$ , **anonymity** of

a user is defined using the following anonymity game  $\text{VABSAnonym}_{\Pi, \mathcal{A}}(\lambda)$ :

1.  $\mathcal{C}$  runs the setup operation and obtains the global setup parameters  $vparams$ .  $\mathcal{C}$  then gives  $1^\lambda$ ,  $n$ ,  $\delta$ , and  $vparams$  to  $\mathcal{A}$ . The time period counter  $t$  is initialized as  $t := 1$ , and is started.
2. At any step,  $\mathcal{A}$  can generate polynomially-many authority keys, but is required to give the public keys to  $\mathcal{C}$ .  $\mathcal{A}$  can also ask  $\mathcal{C}$  to generate authorities. Then,  $\mathcal{C}$  would honestly generate them, and share their public keys with  $\mathcal{A}$ .  $\mathcal{A}$  can generate polynomially-many users under its control or can ask  $\mathcal{C}$  to generate honest users, and has the ability to run for those users  $\text{VABS.UserJoin}$  or  $\text{VABS.AttrIssue}$  with any authority.  $\mathcal{A}$  can demand revocation of any user or any attribute belonging to any user.  $\mathcal{A}$  can adaptively request  $\mathcal{C}$  to sign any message as any user under its control with any attribute policy that the user satisfies. If  $\mathcal{A}$  outputs any string to  $\mathcal{C}$  other than what is explicitly written in protocol steps as expected from  $\mathcal{A}$ , then  $\mathcal{C}$  just ignores it.
3.  $\mathcal{C}$  generates two secret keys  $S_0$  and  $S_1$ .  $\mathcal{C}$  runs  $\text{VABS.UserJoin}$  and  $\text{VABS.AttrIssue}$  for  $s_0$  and  $s_1$  with authorities of  $\mathcal{A}$ 's choice.
4.  $\mathcal{A}$  is given access to the signature oracles of both users:  $\text{VABS.Sign}(\tilde{m}, S_0, \tilde{\beta}, *)$  and  $\text{VABS.Sign}(\tilde{m}, S_1, \tilde{\beta}, *)$ ; where the message  $\tilde{m}$  and the attribute policy  $\tilde{\beta}$  are chosen by  $\mathcal{A}$  as part of its queries. Each oracle stops responding to further queries during that time period when the number of queries reach  $n - 1$ .
5.  $\mathcal{A}$  gives  $\mathcal{C}$  an attribute policy  $\beta$ .  $\mathcal{A}$  gives two messages  $m_0$  and  $m_1$  to  $\mathcal{C}$ .
6.  $\mathcal{C}$  picks a random bit  $b$ .  $\mathcal{C}$  signs both messages as  $s_b$  by running  $\sigma_0 \leftarrow \text{VABS.Sign}(m_0, s_b, \beta, *)$  and  $\sigma_1 \leftarrow \text{VABS.Sign}(m_1, s_{\bar{b}}, \beta, *)$ .  $\mathcal{C}$  then gives  $\sigma_0$  and  $\sigma_1$  to  $\mathcal{A}$ .
7.  $\mathcal{A}$  eventually returns a bit  $b'$ . The output of the game is defined as 1 (i.e.,  $\mathcal{A}$  wins), if  $b = b'$ . Otherwise, the output of the game is defined as 0 (i.e.,  $\mathcal{A}$  loses).

**Definition 6** (Anonymity). A VABS scheme  $\Pi$  provides anonymity, if  $\forall n, \delta \in \text{poly}(\lambda)$ , for each PPT adversary  $\mathcal{A}$ , there exists a negligible function  $n(\cdot)$  such that

$$\Pr[\text{VABSAnonym}_{\mathcal{A}, \Pi}(\lambda) = 1] \leq \frac{1}{2} + n(\lambda)$$

Given a VABS scheme  $\Pi$ , a PPT adversary  $\mathcal{A}$ , a challenger  $\mathcal{C}$ , a unary security parameter  $1^\lambda$ , and a limit  $n$  for a time period duration  $\delta$ , **signature unforgeability** is defined using the following  $\text{VABSForge}_{\mathcal{A}, \Pi}(\lambda)$  game:

1.  $\mathcal{C}$  runs the setup operation and obtains the global setup parameters  $vparams$ .  $\mathcal{C}$  then gives  $1^\lambda$ ,  $n$ ,  $\delta$ , and  $vparams$  to  $\mathcal{A}$ . The time period counter  $t$  is initialized as  $t := 1$ , and is started.
2. At any step,  $\mathcal{A}$  can generate polynomially-many authority keys, but is required to give the public keys to  $\mathcal{C}$ .  $\mathcal{A}$  can also ask  $\mathcal{C}$  to generate authorities. Then,  $\mathcal{C}$  would honestly generate them, and share their public keys with  $\mathcal{A}$ .  $\mathcal{A}$  can generate polynomially-many users under its control or can ask  $\mathcal{C}$  to generate honest users, and has the ability to run for those users  $\text{VABS.UserJoin}$  or  $\text{VABS.AttrIssue}$  with any authority.  $\mathcal{A}$  can demand revocation of any user or any attribute belonging to any user.  $\mathcal{A}$  can adaptively request  $\mathcal{C}$  to sign any message as any user under its control with any attribute policy that the user satisfies. If  $\mathcal{A}$  outputs any string to  $\mathcal{C}$  other than what is explicitly written in protocol steps as expected from  $\mathcal{A}$ , then  $\mathcal{C}$  just ignores it.
3.  $\mathcal{A}$  returns an attribute  $\omega$  to  $\mathcal{C}$ .  $\omega$  must not be queried before as  $\text{VABS.AttrIssue}$  to the authorities under  $\mathcal{C}$ 's control for users under  $\mathcal{A}$ 's control.
4. The game restricts  $\mathcal{A}$  in that  $\mathcal{A}$ 's queries to an attribute authority oracle as  $\text{VABS.AttrIssue}$  with  $\omega$  for users under  $\mathcal{A}$ 's control, the attribute is issued but is *immediately revoked*.
5.  $\mathcal{A}$  eventually generates a message  $m$  (not queried for signing to the users under  $\mathcal{C}$ 's control for an attribute policy  $\beta \wedge \omega$ ) and a VABS  $\sigma$ , where  $\mathcal{A}$  may choose any policy  $\beta$ . The output of the game is defined as 1 (i.e.,  $\mathcal{A}$  wins), if  $\text{VABS.Verify}(m, \sigma, \beta \wedge \omega, \dots) = 1$ . Otherwise, the output of the game is defined as 0 (i.e.,  $\mathcal{A}$  loses).

**Definition 7** (Signature Unforgeability). A VABS scheme  $\Pi$  provides signature unforgeability, if  $\forall n, \delta \in \text{poly}(\lambda)$ , for each PPT adversary  $\mathcal{A}$ , there exists a negligible function  $n(\cdot)$  such that

$$\Pr[\text{VABSForge}_{\mathcal{A}, \Pi}(\lambda) = 1] \leq n(\lambda)$$

Given a VABS scheme  $\Pi$ , a PPT adversary  $\mathcal{A}$ , a challenger  $\mathcal{C}$ , a unary security parameter  $1^\lambda$ , and a limit  $n$  for a time period duration  $\delta$ , **soundness** is defined using the following soundness game  $\text{VABSSound}_{\mathcal{A}, \Pi}(\lambda)$ :

1.  $\mathcal{C}$  runs the setup operation and obtains the global setup parameters  $vparams$ .  $\mathcal{C}$  then gives  $1^\lambda$ ,  $n$ ,  $\delta$ , and  $vparams$  to  $\mathcal{A}$ . The time period counter  $t$  is initialized as  $t := 1$ , and is started.
2. At any step,  $\mathcal{A}$  can generate polynomially-many authority keys, but is required to give the public keys to  $\mathcal{C}$ .  $\mathcal{A}$  can also ask  $\mathcal{C}$  to generate authorities. Then,

$\mathcal{C}$  would honestly generate them, and share their public keys with  $\mathcal{A}$ .  $\mathcal{A}$  can generate polynomially-many users under its control or can ask  $\mathcal{C}$  to generate honest users, and has the ability to run for those users `VABS.UserJoin` or `VABS.AttrIssue` with any authority.  $\mathcal{A}$  can demand revocation of any user or any attribute belonging to any user.  $\mathcal{A}$  can adaptively request  $\mathcal{C}$  to sign any message as any user under its control with any attribute policy that the user satisfies. If  $\mathcal{A}$  outputs any string to  $\mathcal{C}$  other than what is explicitly written in protocol steps as expected from  $\mathcal{A}$ , then  $\mathcal{C}$  just ignores it.

3. The output of the game is defined as 1 (i.e.,  $\mathcal{A}$  wins), if  $\mathcal{A}$  generates at least  $n+1$  message and `VABS` pairs within some time period for the same user  $s$  such that honest executions of the `VABS.Verify` algorithm on at least  $n+1$  of those pairs within the same time period output 1. Otherwise, the output of the game is defined as 0 (i.e.,  $\mathcal{A}$  loses).

**Definition 8** (Soundness). *A VABS scheme  $\Pi$  provides soundness, if  $\forall n, \delta \in \text{poly}(\lambda)$ , for each PPT adversary  $\mathcal{A}$ , there exists a negligible function  $n(\cdot)$  such that*

$$\Pr[\text{VABSSound}_{\mathcal{A}, \Pi}(\lambda) = 1] \leq n(\lambda)$$