Received 7 August 2022, accepted 13 August 2022, date of publication 23 August 2022, date of current version 12 September 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3201071

RESEARCH ARTICLE

VeloCash: Anonymous Decentralized Probabilistic Micropayments With Transferability

TAISEI TAKAHASHI, TAISHI HIGUCHI, AND AKIRA OTSUKA, (Member, IEEE)
Institute of Information Security, Yokohama, Kanagawa 221-0835, Japan

Corresponding author: Taisei Takahashi (dgs194102@iisec.ac.jp)

ABSTRACT Micropayments are one of the challenges in cryptocurrencies. Micropayments on the blockchain have the problem that the fee is high for the transfer amount. As a countermeasure, a method called Layer-two has been proposed to consolidate transactions outside the blockchain and improve the blockchain's throughput. As one of the existing Layer-two schemes, Decentralized Probabilistic Micropayments have been proposed. The winning amount is registered in the blockchain, and the lottery tickets are issued to be won with probability p, which allows us to aggregate approximately $(1/p)$ transactions into one. Unfortunately, existing solutions do not allow for ticket transferability, and the smaller p, the more difficult it is to use them in the real world. Here we propose VeloCash, Decentralized Probabilistic Micropayments with Transferability, which preserves anonymity. By introducing tamper-proof assumptions for sending and receiving the tickets, we make p smaller. As a tamper-proof hardware assumption, VeloCash uses Attested Execution Secure Processors, a formal abstraction of secure processors with attested execution functionality and Direct Anonymous Attestation to achieve anonymity for sending and receiving tickets. VeloCash can detect double-spending attacks perfectly and revoke the adversary's device.

INDEX TERMS Blockchain, anonymity, micropayment, transferability, direct anonymous attestation, attested execution secure processors.

I. INTRODUCTION

Micropayments are minimal payments, e.g., less than \$1, and can be used in a wide range of applications, such as per-page billing in e-books and delivering content billed per minute. However, it is challenging to realize micropayments in the blockchain.

The problems in realizing micropayments in the blockchain are the low throughput and the high blockchain transaction fee. Since the capacity of each block is fixed, miners prioritize transactions that can generate high fees and put off micropayment transactions with low fees. In addition, the blockchain transaction fees do not depend on the amount of money to be transferred. Thus, the blockchain transaction fees can be relatively small for high-value transfers but high for micropayments.

The above problems can be solved by Layer-two [2]. Instead of registering all transactions in the blockchain, Layer-two aggregates small transactions into a few larger ones, increasing transaction throughput and reducing transaction fees. Decentralized probabilistic micropayments [1], [3] have been proposed as one of the methods for Layer-two. It is a lottery-based scheme, the amount of required payments is locked in an escrow, and micropayments are issued as lottery tickets. Let the winning amount be β, and the winning probability is p, the expected value per lottery ticket is $p \cdot \beta$, and the ticket is used as currency. Probabilistic micropayments allow us to aggregate the entire transactions by approximately p. For example, if $10,000$ transactions are processed by a probabilistic micropayments scheme, only $10,000 \cdot p$ transactions will be registered in the blockchain.

Almashaqbeh *et al.* have proposed MicroCash [3] which is a lightweight protocol for non-interactive and sequential payments. The disadvantage of MicroCash is that the game theory guarantees safety against double-spending attacks. Thus, the amount of money for the escrow account, which is confiscated when a double-spending attack is discovered, is considerable. As an example, when $m = 5$ and

The associate editor coordinating the review of this manuscript and approving it for publication was Chien-Ming Chen.

$B_{\text{escrow}} = 2000$, the penalty escrow is $B_{\text{penalty}} = 477.6$. In addition, tickets can only be sent once by the ticket issuer; in other words, the tickets can not be *transferable*.

As MicroCash, when safety is constructed using only a game-theoretic approach, considering penalty escrow, the number of honest users who can receive the ticket, u, is realistically constrained to about 5. If we make u large, we need to make the penalty escrow large in proportion to u. As an alternative plan, if we assume that the users can not commit malicious activity, such as a tamper-proof assumption, u can be large without penalty escrow. However, the smaller p is, the higher the gambling potential becomes and the less the payee can use it for actual economic transactions. If many tickets with a minimal winning probability are sent and do not win, the honest users can not make any income. This is because if the ticket can not be transferable, the payee will not earn any income unless the ticket they received wins. The smaller p, the more the opportunity to get an income is lost.

If the ticket is *transferable*, p can be reduced. The payees do not lose anything since the ticket can be used to pay others even if the ticket is not won. However, it is challenging to achieve transferability with existing solutions. Since the ticket is transferable, the issuer and all users can perform the double-spending attacks. Requiring game-theoretically guaranteed penalty escrow for all users is practically undesirable because of high collateral costs. Suppose the ticket transfer is limited to a tamper-proof device, malicious activities that deviate from the protocol can be prevented, and transferability can be achieved without high penalty escrow.

Takahashi and Otsuka [1] have proposed the probabilistic transferable payment scheme. All the ever-proposed probabilistic schemes are based on lottery tickets where only a small fraction of payees will win the lottery and receive multi-fold awards. Their probabilistic micropayment utilizes transferability, where every payer has to pay a transaction fee proportional to the paid amount, say 10%, and the fees are accumulated inside the transferred ticket. Then, only the winner of the lottery ticket will take all of the accumulated transaction fees as the lottery award. In addition, non-winners will always gain expected revenue with far less speculativity. The value of the ticket will diminish exponentially as transferred until the expected velocity. Most of the payments fall into the range of micropayments.

They assume tamper-proof wallets which prevent double-spending before it happens. Their double-spending detection techniques are shown to detect perfectly when the double-spent ticket is about to be registered in the blockchain (fork detection) and detect probabilistically when received at the payee (collision detection). With these detection techniques, they can eliminate the need for penalty escrow (required in the previous works [3]) and force adversaries to weigh the cost of breaking a tamper-proof wallet against the maximum expected value that the adversary can obtain from the attack.

However, the probabilistic transferable payment scheme proposed by Takahashi and Otsuka [1] does not specially take care of anonymity, and eventually, the transaction history of tickets will be published on the blockchain. Hence its anonymity is equivalent to Bitcoin as far as addresses are not reused.

A. CONTRIBUTION

In this paper, we have achieved stronger notions of anonymity on the previous work of Takahashi and Otsuka [1]. The main contributions of this paper are the followings:

1) Extended anonymity notions for blockchain-based decentralized transferable payment schemes:
   All of the previous anonymity notions [4], [5] assume the existence of the bank as the central authority. Thus, applying those anonymity notions to the blockchain-based decentralized payment schemes is not straightforward. In this paper, we introduced the generalized anonymity notions of transferred electronic cash schemes to cover both centralized and decentralized payment schemes.
2) Revocable anonymity extension for attested execution secure processors:
   Attested Execution Secure Processors (AESP) [6] is the abstraction of tamper-proof secure processors, which enforces every installed program to attach an attested signature as proof to show that the output is the result of the execution of the program. In this paper, we proposed a new mechanism to revoke tampered AESP's utilizing the idea of key extractor when double-spending is detected. In order not to be too abstracted, our key extractor is defined over Enhanced Privacy Identifier (EPID) [7]. Still, our technique can be applied to any Direct Anonymous Attestation (DAA) scheme [8] with Fiat-Shamir proof of knowledge for NP relation.
3) Secure construction of probabilistic anonymous payments with transferability:
   We propose VeloCash, an anonymous probabilistic micropayment scheme with transferability. The construction satisfies all the security and anonymity notions claimed in the theorems.

B. ORGANIZATION OF THIS PAPER

We organize the rest of this paper as follows. Section II describes related works. Section III describes DAA as a preliminary to achieve anonymity. Section IV introduces decentralized probabilistic micropayments with transferability. Section V describes the anonymization of the transferable payment scheme. In Section VI, we prove that VeloCash satisfies the anonymity property and, even with anonymity, VeloCash satisfies the conditions of the transferable payment scheme. In Section VII, we analyze the efficiency of our scheme and compare it with the previous transferable e-cash scheme. Finally, Section VIII concludes this paper.

II. RELATED WORK

A. PAYMENT CHANNELS AND NETWORKS

The payment channel establishes a private, peer-to-peer transmission protocol. Based on pre-defined rules, two parties can agree to update their state and transfer money by exchanging authenticated state transitions in a so-called *off-chain* fashion.

To conduct a transaction on *Payment Channel*, two parties must first register a shared 2-of-2 multi-sig escrow fund in the blockchain and establish the channel. The payment channel enables the two parties to perform transactions through private communications. After the sending and receiving are completed in the channel, the final fixed value is registered in the blockchain. Only two transactions are registered in the blockchain per channel, the escrow fund transaction and the final fixed value. A payer can send money to a user who has not established a channel with the payer through the *Payment Network* between users who have established a channel. For example, suppose Alice sends 0.1 coins to Charlie, who has not established a channel with Alice. First, Alice sends 0.1 coins to Bob, with whom Alice has a channel. Next, Bob sends 0.1 coins to Charlie, whom Bob has a channel.

Unfortunately, the payment channel and the network have the disadvantage of high collateral cost [9]. Each time a channel is established, escrow is required between two parties. Also, the longer the payment network path, the more reserves are required and locked. Since the reserves can not be used during the locktime periods, the reserves represent a lost opportunity. Furthermore, in a payment network, a fee is charged for each pass through the nodes. Therefore, it is impractical to adopt a payment network for micropayments since it is undesirable to incur the cost for each node.

B. PROBABILISTIC MICROPAYMENTS

The idea of Probabilistic Micropayments has been proposed by Wheeler [10] and Rivest [11]. Since small payments would be costly if settled each time, they proposed a lottery-style protocol where the ticket issuer deposits a large amount of money in the bank, and the winner could receive the money if they won. The lottery tickets can be used as currency, and the value per ticket is regarded as the ticket's expected value. In this scheme, the existence of a bank is mandatory, and participants are limited to people who have a relationship with the bank.

MICROPAY [12] and DAM [13] have been proposed as Decentralized Probabilistic Micropayments using blockchain. Since both have a large overhead supporting sequential micropayments, Almashaqbeh *et al.* have proposed MicroCash [3] which is a light-weight protocol for non-interactive and sequential payments.

C. INTERMEDIATED ANONYMITY

In TumbleBit, an untrusted intermediary, *Tumbler*, exists between Alice (the payer) and Bob (the payee), and even Tumbler can not link the payer and the payee.

Zhang *et al.* [14] have extended TumbleBit and proposed an Anonymous Off-blockchain Micropayments scheme, AOM. In AOM, an escrow for micropayments under the standard RSA assumption is set up by Alice with an intermediary, who later aggregates the transactions. Ferretti *et al.* [15] have introduced Bitcoin's *Pay-To-Script-Hash (P2SH)* on TumbleBit, allowing anonymous transfer of coins from Bob to a third party Charlie.

Both [14], [15] realize the anonymous coin spending through an intermediary. Their schemes require setting up an online intermediary for each spending, possibly increasing the transaction fee, which is unsuitable for micropayments.

D. E-CASH

If lottery tickets are used as currency, it is desirable to have anonymity as physical currency has. Several schemes for anonymous transfer have been proposed earlier. However, some problems were pointed out, such as the existence of the trusted party. Recently, Foteini Baldimtsi *et al.* proposed a new anonymous transferable scheme, Transferable E-cash [16]. It makes it possible to create a fully anonymous transferable electronic cash scheme without trusted third parties with malleable signatures to construct their secure and anonymous transfer of coins. However, the Transferable E-cash scheme has the drawbacks such as efficiency pointed out by Bauer *et al.* [5]. In the work [5] they revisited and proposed the definition of the security and anonymity properties, which Transferable E-cash systems should satisfy - (1) unforgeability: an adversarial user cannot spend more e-coins than he withdrew, and (2) anonymity: nobody can link spending transactions to each other or to specific withdrawal transactions. Their definition of anonymity consists of three parts - (1) coin anonymity, (2) user anonymity and (3) coin transparency.

III. PRELIMINARIES

As in [1], the transferable payment scheme is realized using tamper-proof devices. In this study, which realizes the anonymity of the transferable payment scheme, we also assume tamper-proof devices. More specifically, we use Attested Execution Secure Processors (AESP), that abstracts ''attested execution'' secure processors. In the ''attested execution'' of AESP, the output of the installed program is digitally signed with the secret key in the AESP. The signature enables verification that the output is indeed from a legitimate AESP.

Direct Anonymous Attestation (DAA) has been used for the attested signature. DAA can be seen as group signatures [17] but differs from group signatures in that DAA does not have an *opening* algorithm that allows the group manager to obtain the identity of the signer from the signature. Instead of having *opening* function, DAA has a so-called ''revocation function''. Suppose a particular hardware module has been broken and its secret key has been compromised; the secret key is placed on the revocation list. When a verifier receives the signature, he can verify whether it is signed with the secret key on the revocation list.

In this study, we use AESP to send and receive tickets, and DAA is used for attested signatures and for sending tickets. There are three reasons to adopt DAA: 1) For anonymization of attested signature. 2) To extract the adversary's secret if the double-spending attacks are performed 3) To revoke the extracted adversary's secret key. In particular, extracting and revoking the adversary's secret key is a strong motivation for adopting DAA. We adopt Enhanced Privacy ID (EPID), a DAA scheme that uses Fiat-Shamir proof of knowledge for NP relation. Thus, since the Extractor can be configured, it is possible to extract and revoke the adversary's secret key from the double-spending tickets with the same commitment but different challenges.

A possible alternative to anonymization other than DAA is using ring signatures. Traceable ring signatures [18], [19] limit the number of times a secret key can sign transactions, making it possible to detect double-spending attacks. However, ring signatures grow proportionally to the size of the anonymity set. There is an inevitable trade-off between anonymity and signature size while DAA signature size is constant.

A. DIRECT ANONYMOUS ATTESTATION (DAA)

Direct Anonymous Attestation (DAA) remote authentication scheme for trusted hardware module has been proposed by Brickell *et al.* [8]. DAA has been adopted by the Trusted Computing Group (TCG).

There are multiple DAA schemes have been proposed [20], [21], [22], [23], [24]. In this paper, we adopt *Enhanced Privacy ID (EPID)* scheme proposed by Brickell and Li [7], [25]. EPID is a scheme proposed by Intel Corporation and is already in use in the real world, embedded in chipsets such as Intel SGX. EPID is compliant with International Standards Organization standard ISO/IEC 20008, 20009 and approved by the Trusted Computing Group (TCG) as the recommended scheme. Intel has made EPID an open-source to processor manufacturers under the Apache 2 license. In 2015, Microchip and Atmel announced that they had licensed the EPID technology [26], [27].

1) SPECIFICATION OF EPID

EPID is one of the DAA schemes which preserves anonymity proposed by Brickell and Li [7], [25]. There are two types of revocation in EPID, the secret key based revocation, and the signature based revocation. In the secret key based revocation, put the secret key into the secret (private) key based revocation list Priv-RL. In the signature based revocation, put the signature into the revocation list Sig-RL. The verification process will invalidate the signatures with the keys in the corresponding revocation lists in both revocation schemes.

There are four entities in EPID: an issuer $\mathcal{I}$, a revocation manager $\mathcal{R}$, platforms[1] $\mathcal{P}_i$, and verifiers $\mathcal{V}$. The scheme consists of five polynomial-time algorithms:

$$\Pi_{\mathrm{EPID}} = \left(\mathsf{Setup}, \mathsf{Join}, \mathsf{Sign}, \mathsf{Com}, \mathsf{Verify}, \mathsf{Revoke}\right). \quad (1)$$

See Appendix A for more details.

[1]In the original definition by Brickwell and Li [7], [25], platforms are the secure hardware-based signing entities such as SGX in Intel processors. Pass *et al.* [6] refers to the signing entities to produce attested signatures as ideal functionality $\mathcal{G}_{\mathrm{att}}$, and they used the term 'platform' for the entities who is allowed to invoke functionalities in $\mathcal{G}_{\mathrm{att}}$.

2) SECURITY DEFINITION OF EPID

An EPID scheme is secure if it satisfies the following three requirements: correctness, anonymity, unforgeability [7], [8].

The correctness requires that every signature a platform generates is valid except when the platform is revoked by the secret key based revocation or the signature based revocation.

Theorem 1 (Theorem 4 of [28]): The EPID scheme is correct.

The EPID scheme is anonymous if the adversary can not determine from the signature the secret key used to generate the signature.

Theorem 2 (Theorem 5 of [28]): An EPID scheme is anonymous in the random oracle model under the decisional Diffie-Hellman assumption in G_3.

The EPID scheme is unforgeable if the adversary can not forge a valid signature with all secret keys known to the adversary that has been revoked.

Theorem 3 (Theorem 6 of [28]): The EPID scheme is unforgeable in the random oracle model under the strong Diffie-Hellman assumption in (G_1, G_2).

See Appendix C for more detailed definitions and Appendix B for the construction of EPID.

IV. PROBABILISTIC TRANSFERABLE PAYMENT SCHEMES

A. PROBABILISTIC TRANSFER SCHEME

The probabilistic transferable payment scheme [1] uses the lottery-based mechanism to enable micropayments. The ticket issuer creates an escrow account with the winning amount and registers it on the blockchain. Then, the ticket is issued based on the escrow account and distributed among users. The users who receive a ticket that meets the winning condition can receive the winning amount.

All tickets are sent and received via a tamper-proof device wallet, and tickets are sent using the digital signature key in the wallet. Since the tamper-proof device is used, double-spending attacks would not be performed. However, in reality, tamper-proof devices can be broken, and double-spending attacks can be performed. Thus, Takahashi and Otsuka [1] proposed two detection methods that can completely detect double-spending attacks and place an upper bound on the expected value of the attack when the tamper-proof devices are broken.

We denote by σ_A a signature signed by A.

Definition 1: A lottery tickets τ consists of a four-tuple:

$$\left(A, B, \tau_{pre}, \sigma\right) \quad (2)$$

where A and B are accounts of a sender and a receiver, respectively. τ_{pre} is a reference to the previous ticket or the escrow account ϵ. σ is the EPID digital signature on τ.

For readability, we write a ticket τ as:

$$\tau = \left(A \rightarrow B, \tau_{\mathrm{pre}}\right)_X . \quad (3)$$

Let A and B both be the receiving addresses. Also, X is the signature of the sender A, which indicates the signer's identity.

We define $|\tau|$ the "number of generations" of τ, which is the length of the sequence from ϵ to τ. For example, $|\tau| = n$ if there exists a sequence $\tau_1, \ldots, \tau_{n-1}$ such that $\tau_\epsilon \prec \tau_1 \prec \tau_2 \prec \cdots \prec \tau_{n-1} \prec \tau$. We define $|\tau| = \infty$ if no such sequence exists.[2] To write compactly, we denote by τ_i the i-th generation of τ.

Definition 2 (Transferred Transaction): Two tickets $\tau_i = \left(A \to B, \tau_{pre}\right)_X$ *and* $\tau_{i+1} = \left(A' \to B', \tau'_{pre}\right)_{X'}$ *are said to be* transferred *if and only if following properties satisfies*:

$$\begin{cases} \mathsf{Hash}(\tau_i) = \tau'_{pre} \\ A = X,\ B = A' = X' \\ \sigma \textit{ is valid} \end{cases} \tag{4}$$

Then, we write $\tau_i \prec \tau_{i+1}$.

We write $\tau_i \lll \tau_{i+n}$ if there exists a sequence of ordered lottery tickets $\tau'_1 \prec \ldots \prec \tau'_n$ for $n \geq 1$ and they satisfy $\tau_i \prec \tau'_1$ and $\tau'_n \prec \tau_{i+n}$. If τ has no previous lottery tickets, the ticket is called a "genesis" ticket. For the genesis tickets τ_1 tied to an escrow account ϵ, we specially denote by $\epsilon \prec \tau_1$ so that lottery tickets are simply written as:

$$\epsilon \prec \tau_1 \prec \tau_2 \prec \ldots \prec \tau_n. \tag{5}$$

Definition 3: A lottery ticket τ *is said to be* valid *with respect to a blockchain* $\mathbb{C}$ *for some security parameter* k *if and only if there exists an escrow account* ϵ *and a sequence of transactions* $\{\tau_1, \ldots, \tau_n\}$ *such that*

$$\epsilon \in \mathbb{C}^{\lceil k} \quad \textit{and} \quad \epsilon \prec \tau_1 \prec \ldots \prec \tau_n \prec \tau. \tag{6}$$

$\mathbb{C}^{\lceil k}$ denotes the set of blocks that are k or more before the beginning of the blockchain. This notion is borrowed from Garay *et al.* [29].

To specify the parameters of the transferable transaction, the escrow account ϵ further contains:

$$(\beta, p, q, \mu) \tag{7}$$

where β is the ticket winning amount, p is the probability for determining winning a ticket, q is the lottery ticket transaction rate of proportional fee scheme,[3] and μ is a fixed value used to determine the winning ticket.

We define an escrow as *active* when the ticket generated from the escrow has been distributed. Still, the ticket does not meet the condition for winning and is not registered in the blockchain.

Definition 4 (Active Escrow): We define escrow account ϵ *to be* active *if and only if the following conditions are met*:

$$\begin{cases} \tau \quad \textit{is } \mathsf{valid} \\ \epsilon \in \mathbb{C}^{\lceil k} \wedge \tau \notin \mathbb{C}^{\lceil k} \end{cases} \tag{8}$$

where τ *is the sequence of transactions such as* $\{\epsilon \prec \tau_1 \prec \cdots \prec \tau_n\}$.

[2]For practical purposes, we assume that the height of τ can only be measured when all tickets in the sequence from τ_ϵ to τ are given. Even if such a sequence exists, the height of τ is considered to be ∞ unless the entire sequence is specifically presented.

[3]See Appendix D.

1) TICKET WINNING CONDITION

This section describes the design of the ticket winnings.

Definition 5: τ_v *is said to be* win *if and only if the following properties satisfy*:

$$\mathsf{win} = \left\{\tau_v \;\middle|\; \mathrm{VDF}\left(h_0 + v \cdot \mu\right) < D, v \in \mathbb{N}\right\} \tag{9}$$

where v *is the number of generations of* τ, h_0 *is the block height containing the escrow account and* μ *is the fixed number specified in the escrow account* ϵ.

The probability p is calculated using a simple Verifiable Delay Function (VDF) [30]. The calculation can be done after a certain period of time has elapsed from when the ticket is transferred according to the number of generations. For example, if a ticket with $h_0 = 100$, $\mu = 5$ and $v = 3$ is received, the VDF value will be known when the block height of 115 is confirmed.

If the ticket $\tau \in \mathsf{win}$ has already been sent and the ownership has been transferred, the user with the latest ownership is considered eligible and can get the winning amount β.

Definition 6: τ_v *is said to be* eligible *if and only if the following properties satisfies*:

$$\mathsf{eligible} = \left\{\tau_v \;\middle|\; {}^{\exists}\tau_{v'} \in \mathsf{win} \wedge \tau_{v'} \lll \tau_v\right\} \tag{10}$$

eligible ticket will be considered as the final winning ticket. Thus, the user who has the eligible ticket can get β from the escrow account ϵ.

2) PROPORTIONAL FEE SCHEME

Proportional Fee Scheme is a payment scheme first introduced by Takahashi and Otsuka [1]. This scheme is where each time a payer transfers a ticket; the payer bears the fee based on the number of generations of the ticket.

Definition 7 (Proportional Fee Scheme): Let q *be the lottery ticket transaction fee rate. Suppose a payer sends a ticket* τ_i*, and the payee gives goods or services worth* $(1-q)^i\beta$ *to the sender. The fee borne by each payment is* $(1-q)^{i-1}q\beta$.

This scheme has the advantage that the payment fee can be smaller than the blockchain transaction fee. The average transaction fee for cryptocurrencies, especially Bitcoin, is approximate \$2 [31]. By contrast, in our transferable scheme, let $\beta = \$100$, $p = \frac{1}{100}$ and $q = \frac{1}{10}$, the fee for a \$1 transfer is about 10 cents.

Definition 8 (Condition of Proportional Fee Scheme): In transferable payment, Proportional Fee Scheme is achieved if and only if the ticket's series of transactions from ϵ *is referenceable.*

3) DOUBLE-SPENDING ATTACKS DETECTION METHODS

The security design requires the adversary to have an expected value obtained from the attack that exceeds the cost by breaking the tamper-proof device.

Fork detection is a method that perfectly detects double-spending attacks. *Fork* implies that two tickets are assumed to exist, and the resulting ticket prefixes are identical except for k elements from both ends of the ticket.

Definition 9 (Fork of Transferred Transactions): Given two series of transactions initiated with the same escrow account $\tau = \{\epsilon \prec \tau_1 \prec \cdots \prec \tau_n\}$, *and* $\tilde{\tau} = \{\tilde{\epsilon} \prec \tilde{\tau}_1 \prec \cdots \prec \tilde{\tau}_{n'}\}$, *the series of transactions are said to be* Fork *if and only if it satisfies:* $^{\exists}k \in \mathbb{Z}_{\geq 0}$,

$$\begin{cases} \epsilon = \tilde{\epsilon} \\ \tau^{\lceil k} \lll \tilde{\tau} \wedge \tilde{\tau}^{\lceil k} \lll \tau \\ \tau^{\lceil k-1} \not\lll \tilde{\tau} \vee \tilde{\tau}^{\lceil k-1} \not\lll \tau. \end{cases} \tag{11}$$

Lemma 1 (Fork Detection): Given two series of transactions $(\tau, \tilde{\tau}) \in$ Fork, *there exists an efficient fork detection algorithm that outputs the double-spending transactions.*

Every time one of the Fork transactions is published on the blockchain, players holding the other Fork transactions can identify the double-spending transaction perfectly using the above-described procedure. The adversary's address will be revealed as the common spender of the two resulting transactions in Lemma 1.

The following *collision detection* upper-bounds the adversary's expected utility.

Definition 10 (Collision of Transferred Transactions): Suppose each user has multiple addresses. It is said to be a collision if and only if there exists a user who receives two transactions τ *and* $\tilde{\tau}$ *such that* $(\tau, \tilde{\tau}) \in$ Fork *in any of his addresses.*

Further, to work the collision detection effectively, we assume that the ticket transaction protocol is conducted in the three rounds: Round 1) The adversary sends the tickets to an honest payee. Round 2) The payee checks received tickets for the collisions. If the payee finds *collision*, he broadcasts the tickets, say τ and $\tilde{\tau}$, in the collision (or publishes them on the blockchain). Round 3) If the collision is not detected, the payee gives products or services to the payer in return. However, if the collision is detected, the adversary's address is rejected and will never be accepted by all honest users.

Theorem 4 (Collision Detection): Let u *be the number of users participating in the transfer scheme, where each user has* $\alpha\,(\geq\ 2)$ *addresses. By collision detection in the round-based synchronous network, the expected utility of double-spending attack* $\mathbb{E}_d$ *is upper-bounded by the following inequality*:

$$\mathbb{E}_d \leq \sqrt{\frac{u}{e}}\beta. \tag{12}$$

By the fork and collision detection methods, the adversary can not profit under the following conditions:

$$\sqrt{\frac{u}{e}}\beta < \Phi \tag{13}$$

where Φ is the cost of breaking κ-tamper proof wallet.

As an example, consider the maximum expected utility value $\mathbb{E}_d$ with $u = 1,000,000$ and $\beta = \$100$. Applying the equation (12) produces $\mathbb{E}_d \lessapprox \$60,700$.

See formal definitions and proofs in Appendix D.

V. ANONYMOUS PAYMENT SCHEMES

The results in the preceding section show that the cost of breaking a κ-tamper proof wallet must exceed the expected utility of an adversary. In this section, we follow the formal abstraction of attested execution secure processors [6] with adequate tamper-resistance (whose breaking cost exceeds Φ).

All previously proposed anonymous and transferable electronic cash schemes [4], [5], [16], [32], [33] assume that:

1) the existence of central authority (bank), and
2) only the bank can detect double-spending.

Our decentralized blockchain-based transferable payment scheme is described in the previous section; however, every player is eligible to set up an escrow account and initiate offline transferable payments; hence no central authority (bank) exists. Further, our collision-detection and fork-detection techniques enable every recipient of transferable payments to detect double-spending and publish the evidence on the blockchain. That is, every player can potentially detect double-spending. Finally, to capture the anonymity in these decentralized settings, we will define generalized anonymity notions in the following subsections.

ALGORITHMS (CG08)

A conventional transferable e-cash system generally involves two types of players: a bank $\mathcal{B}$ and a user $\mathcal{U}$. Whereas a blockchain-based transferable e-cash system has no banks, a blockchain $\mathbb{C}$ takes the role of a bank $\mathcal{B}$. $\mathbb{C}$ can be regarded as $\mathcal{B}$ that holds no secret information and publishes all of its views and the deposited coin list $\mathcal{L}$.

A coin is represented by an identifier id and some values π needed to prove its validity.

- ParamGen(1^λ) is a probabilistic algorithm that outputs the system's parameters param (including the security parameter λ). We assume all functions take param as their inputs unless otherwise specified.
  Note: param may contain the genesis block of the blockchain $\mathbb{C}$. For schemes assuming DAA-based anonymous signature schemes with tamper-proof devices, ParamGen generates the DAA public key and secret key pair (gpk, gsk) and param contains gpk for the verification of anonymous signatures.
- BKeyGen(param) (resp. UKeyGen(param)) is a probabilistic algorithm executed by $\mathcal{B}$(resp. $\mathcal{U}$) that outputs the key pair $(\mathsf{sk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{B}})$ (resp. $(\mathsf{sk}_{\mathcal{U}}, \mathsf{pk}_{\mathcal{U}})$).
  Remark: Blockchain-based transferable e-cash systems have the blockchain $\mathbb{C}$ *in place of the bank* B, *and* $\mathbb{C}$ *has no secret information. Thus,* BKeyGen(param) *outputs nothing such that* $(\mathsf{sk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{B}}) = (\perp, \perp)$.
  Remark: For schemes assuming DAA-based anonymous signature schemes with a tamper-proof device, UKeyGen(param) *invokes* Join(gpk) *protocol with the manufacturer who holds* gsk, *and stores* sk_i *securely in the tamper-proof device within a platform* $\mathcal{P}_i$. *Such systems share the group public key* gpk, *and no individual*

public key pk_i *for the platform* $\mathcal{P}_i$ *exists. Hence,* $(\mathsf{sk}_{\mathcal{U}}, \mathsf{pk}_{\mathcal{U}}) = (\mathsf{sk}_i, \perp)$.

- $\mathsf{Withdraw}\left(\mathcal{B}\left(\mathsf{sk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{U}}\right), \mathcal{U}\left(\mathsf{sk}_{\mathcal{U}}, \mathsf{pk}_{\mathcal{U}}, \mathsf{pk}_{\mathcal{B}}\right)\right)$ is an interactive protocol where $\mathcal{U}$ withdraws from $\mathcal{B}$ one coin. At the end, $\mathcal{U}$ either gets a coin $C = (\mathsf{id}, \pi)$ and outputs OK, or outputs $\perp$. The output of $\mathcal{B}$ is either its view $\mathcal{V}^{\mathrm{W}}_{\mathcal{B}}$ of the protocol (including $\mathsf{pk}_{\mathcal{U}}$), or $\perp$.
  Remark: $\mathcal{V}^{\mathrm{W}}_{\mathcal{B}}$ *including* $\mathsf{pk}_{\mathcal{U}}$ *is published on the blockchain* $\mathbb{C}$ *in blockchain-based systems. The same amount of the coin* C *is funded in an address* ϵ *on the blockchain* $\mathbb{C}$*, where* ϵ *is related to the public key* $\mathsf{pk}_{\mathcal{U}}$ *and spendable using the secret key* $\mathsf{sk}_{\mathcal{U}}$*.*
- $\mathsf{Spend}\left(\mathcal{U}_j(\mathsf{id}, \pi, \mathsf{pk}_{\mathcal{U}_i}), \mathcal{U}_i\left(\mathsf{sk}_{\mathcal{U}_i}, \mathsf{pk}_{\mathcal{B}}\right)\right)$ is an interactive protocol where $\mathcal{U}_j$ gives a coin to $\mathcal{U}_i$. At the end, either $\mathcal{U}_i$ outputs a coin $C = (\mathsf{id}_C, \pi_C)$ or $\perp$, and either $\mathcal{U}_i$ saves that C is a spent coin and outputs OK, or $\mathcal{U}_i$ outputs $\perp$.
- $\mathsf{Deposit}\left(\mathcal{U}\left(\mathsf{id}, \pi, \mathsf{sk}_{\mathcal{U}}, \mathsf{pk}_{\mathcal{U}}, \mathsf{pk}_{\mathcal{B}}\right), \mathcal{B}\left(\mathsf{sk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{U}}, \mathcal{L}\right)\right)$ is an interactive protocol where $\mathcal{U}$ deposits a coin (id, π) at the bank $\mathcal{B}$. If (id, π) is not consistent/fresh, then $\mathcal{B}$ outputs $\perp_1$. Else, if id already belongs to the list of spent coins $\mathcal{L}$, then there is an entry $\left(\mathsf{id}, \pi'\right)$ and $\mathcal{B}$ outputs $\left(\perp_2, \mathsf{id}, \pi, \pi'\right)$. Else, $\mathcal{B}$ adds (id, π) to its list $\mathcal{L}$, credits $\mathcal{U}$'s account, and returns $\mathcal{L}$. $\mathcal{U}'$'s output is OK or $\perp$.
  Remark: Blockchain-based transferable e-cash systems have the blockchain $\mathbb{C}$ *in place of the bank* B*. In these systems,* $\mathsf{Deposit}$ *is conducted as a transfer of money from the address* ϵ *to the address of* $\mathcal{U}$ *if the coin* (id, π) *is consistent and the money in the address* ϵ *has never spent.*
  Remark: In the blockchain-based transferable e-cash systems, $\mathcal{L}$ *is a part of the blockchain* $\mathbb{C}$*. If* id *already belongs to the blockchain* $\mathbb{C}$*, or its list of spent coin* $\mathcal{L}$*, then there is an entry* (id, π') *and every user* $\mathcal{U}'$ *who received* (id, π) *can output* $(\perp_2, \mathsf{id}, \pi, \pi')$*. Else,* (id, π) *is published on the blockchain* $\mathbb{C}$ *and added to its list of spent coins* $\mathcal{L}$*.* $\mathcal{U}$*'s output is* OK *or* $\perp$*.*
- $\mathsf{Identify}\left(\mathsf{id}, \pi, \pi'\right)$ is a deterministic algorithm executed by $\mathcal{B}$ that outputs a public key $\mathsf{pk}_{\mathcal{U}}$ and a proof Π_G.
  Remark: In the blockchain-based transferable e-cash systems, $\mathsf{Identify}\left(\mathsf{id}, \pi, \pi'\right)$ *is a deterministic algorithm executed by every user* $\mathcal{U}$ *that outputs a double-spender's secret key* sk_i *since DAA-based anonymous signature schemes have no individual public keys* pk_i*. The systems with such anonymous schemes can not output* $\mathsf{pk}_{\mathcal{U}}$ *of the double spender. Therefore, in such systems,* $\mathsf{Identify}\left(\mathsf{id}, \pi, \pi'\right)$ *outputs* $(\perp, \Pi_G)$*. We consider that* Π_G *includes the double-spenders secret keys* sk_i *and the following* $\mathsf{VerifyGuilt}((\mathsf{id}, \pi), \Pi_G)$ *determines whether the coin* (id, π) *is spent by a guilty user or not.*
- $\mathsf{VerifyGuilt}\left(\mathsf{pk}_{\mathcal{U}} \text{ or } (\mathsf{id}, \pi), \Pi_G\right)$ is a deterministic algorithm that can be executed by any players. It outputs 1 if Π_G is correct and 0 otherwise.
  Remark: In the blockchain-based transferable e-cash systems, we assume Π_G *is published on the blockchain.*

A. ORACLES

Our security games use oracles. We adopt the oracle notions from [4], [5].

1) GLOBAL VARIABLES

We store all information about users in the user list $\mathcal{UL}$ consisting of $\mathcal{U}_i = (\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{uds}_i)$ where uds_i indicates how many times user $\mathcal{U}_i$ has performed the double-spending attacks. The set of supplied coins by the oracles is denoted by $\mathcal{SC}$, and the set of all coins owned by the oracles is denoted by $\mathcal{OC}$. The set of deposited coins is denoted by $\mathcal{DC}$.

If an error occurs during the execution of the oracles, the oracles output $\perp$. Otherwise, the call of the oracles is assumed to have succeeded.

a: REGISTRATION AND CORRUPTION USERS

The adversary can instruct the creation of honest users or passively observe registration. It can moreover *spy* on users, meaning that the adversary can learn the user's secret key. Note that the *spy* can not be performed on the challenge users.

- `BRegist`() plays the bank side of the key generation algorithm $\mathsf{BKeyGen}$ and interacts with the adversary. If successful, it adds $(\mathsf{pk}, \perp, \mathsf{uds} = 0)$ to $\mathcal{UL}$.
  Remark: Since blockchain-based transferable e-cash systems have the blockchain $\mathbb{C}$ *in place of the bank and* $\mathsf{BKeyGen}$ *outputs nothing, oracle* `BRegist`() *outputs nothing.*
- `URegist`() plays the user side of the key generation algorithm $\mathsf{UKeyGen}$ when the adversary controls the bank. Upon successful execution, it adds $(\mathsf{pk}, \mathsf{sk}, 0)$ to $\mathcal{UL}$.
  Remark: In the blockchain-based transferable e-cash systems, `URegist`() *plays when the issuer* $\mathcal{I}$ *of DAA-based anonymous signature schemes are controlled by the adversary. Upon successful execution, it adds* $(\perp, \mathsf{sk}, 0)$ *to* $\mathcal{UL}$*.*
- `Regist`() plays both parties in the $\mathsf{BKeyGen}$ and $\mathsf{UKeyGen}$ algorithm and adds $(\mathsf{pk}, \mathsf{sk}, 0)$ to $\mathcal{UL}$.
  Remark: Since blockchain-based transferable e-cash systems have the blockchain $\mathbb{C}$ *in place of the bank and* $\mathsf{BKeyGen}$ *outputs nothing, the oracle* `Regist`() *outputs nothing.*
- `Spy`(i), for $i \leq |\mathcal{UL}|$, returns the user i's secret key sk_i.

b: WITHDRAWAL ORACLES

The adversary can withdraw a coin from the bank or passively observe a withdrawal.

- `BWith`(i) plays the bank side of $\mathsf{Withdraw}(\mathcal{B}, \mathcal{U}_i)$ algorithm. This oracle updates $\mathcal{SC}$ by adding $\mathcal{V}^{\mathrm{W}}_{\mathcal{B}}$ with bit 1 to flag it as a corrupted coin.

Remark: Since blockchain-based transferable e-cash systems have the blockchain $\mathbb{C}$ in place of the bank, oracle $\texttt{BWith}(i)$ *outputs nothing.*

- $\texttt{UWith}(i)$ plays the user side $\mathsf{Withdraw}(\mathcal{B}, \mathcal{U}_i)$ when the adversary controls the bank. This oracle updates $\mathcal{OC}$ by adding the value $(\mathcal{U}_i, \mathsf{id}, \pi)$.
  Remark: Since blockchain-based transferable e-cash systems have the blockchain $\mathbb{C}$ in place of the bank, $\texttt{UWith}(i)$ *plays with* $\mathsf{Withdraw}(\perp, \mathcal{U}_i)$ *when the adversary controls the issuer. Then, it updates $\mathcal{OC}$ by adding the value $(\mathcal{U}_i, \mathsf{id}, \pi)$.*
- $\texttt{With}(i)$ simulates $\mathsf{Withdraw}$ protocol execution of both $\mathcal{B}$ and user $\mathcal{U}_i$, and it updates $\mathcal{OC}$ as for $\mathsf{Withdraw}(\mathcal{B}, \mathcal{U}_i)$ and $\mathcal{SC}$ by adding $\mathcal{V}^{\mathrm{W}}_{\mathcal{B}}$ with flag 0.
  Remark: Since blockchain-based transferable e-cash systems have the blockchain $\mathbb{C}$ in place of the bank, oracle $\texttt{With}(i)$ *outputs nothing.*

c: SPEND AND DEPOSIT ORACLES

- $\texttt{Spd}(j)$ plays the role of user $\mathcal{U}_j$ by spending a coin in $\mathcal{OC}$ owned by user $\mathcal{U}_j$. It uses and updates the entry $(\mathcal{U}_j, \mathsf{id}, \pi)$ of $\mathcal{OC}$ as the Spend protocol.
- $\texttt{Rcv}(i)$ makes honest user i receive a coin from the adversary and updates the set of $\mathcal{OC}$ by adding a new entry $(\mathcal{U}_i, \mathsf{id}, \pi)$.
- $\texttt{S\&R}(j, i)$ plays the role of both $\mathcal{U}_j$ and $\mathcal{U}_i$ and it executes the spending of a coin owned by $\mathcal{U}_j$ to user $\mathcal{U}_i$. It updates $\mathcal{OC}$ by adding the value $(\mathcal{U}_i, \mathsf{id}, \pi)$ and by flagging the coin as spent by $\mathcal{U}_j$.
- $\texttt{BDepo}()$ lets $\mathcal{A}$ deposit a coin. It runs $\mathcal{U}_i$ in $\mathsf{Deposit}$ using the bank's secret key $\mathsf{sk}_{\mathcal{B}}$. If successful, it adds the received coin to $\mathcal{DC}$ or runs $\mathsf{Identify}$ and returns $(\mathsf{pk}, \Pi_G) \leftarrow \mathsf{Identify}(\mathsf{id}, \pi, \pi')$.
  Remark: Since blockchain-based transferable e-cash systems have the blockchain $\mathbb{C}$ in place of the bank, oracle $\texttt{BDepo}()$ *outputs nothing.*
- $\texttt{Depo}(j)$, the honest deposit oracle, runs $\mathsf{Deposit}$ between the j-th coin owner in $\mathcal{OC}$ and an honest bank. If successful, it adds the received coin to $\mathcal{DC}$ or runs $\mathsf{Identify}$ and returns $(\mathsf{pk}, \Pi_G) \leftarrow \mathsf{Identify}(\mathsf{id}, \pi, \pi')$.
  Remark: Since blockchain-based transferable e-cash systems have the blockchain $\mathbb{C}$ in place of the bank, $\texttt{Depo}(i)$ *plays the role of the user $\mathcal{U}_i$ during a* $\mathsf{Deposit}$ *algorithm.*

B. SECURITY NOTIONS

In this section, we discuss the security properties. To achieve anonymity in Takahashi and Otsuka [1], we need to satisfy two major properties.

The first one is Security properties (*Economic properties* and *Security properties*). For achieving anonymity, we borrow the definition of Transferable E-Cash from Bauer *et al.* [5].

The second one is *Double-spending attacks Detection methods* and *Proportional Fee Scheme*. In addition to satisfying anonymity, we outline Double-spending attacks Detection methods, which are the core security design of [1], and organize the conditions that must be satisfied.

```
Experiment: $\mathsf{Expt}^{\mathtt{sound}}_{\mathcal{A},\Pi}(\lambda)$
1 : $\mathsf{param} \leftarrow \mathsf{ParamGen}(1^\lambda)$; $\mathsf{pk}_{\mathcal{B}} \leftarrow \mathcal{A}(\mathsf{param})$
2 : $(b, i_1, i_2) \leftarrow \mathcal{A}^{\mathtt{URegist},\mathtt{Spy}}$
3 : if $b = 0$ then run $\texttt{UWith}(i_1)$ with $\mathcal{A}$
4 : else run $\texttt{Rcv}(i_1)$ with $\mathcal{A}$
5 : return 0 if this outputs $\perp$
6 : run $\texttt{S\&R}(i_1, i_2)$;
7 : if one party outputs $\perp$
8 :     return 1
9 : else return 0
```

FIGURE 1. Game for ***soundness.***

Economic properties ensure that users do not incur economic losses when they participate in the system. It consists of *soundness*, *unforgeability*, and *exculpability*.

We define a negligible function negl if for every positive polynomial function p there exists $\lambda_0 \in \mathbb{N}$ such that for all $\lambda > \lambda_0$ the following holds:

$$\mathsf{negl}(\lambda) < \frac{1}{p(\lambda)}. \tag{14}$$

1) SOUNDNESS

Suppose an honest user issues or accepts a ticket during a transfer; he should be guaranteed that others will accept the ticket. In that case, either honest users when transferring or the blockchain escrow account when claiming.

The game is formalized in Figure 1. The adversary issues a ticket or sends the received ticket τ to the user i_0. If the result of sending the ticket to the user i_1 or claiming to the blockchain is false, the adversary wins.

Definition 11 (Soundness): An anonymous transferable scheme is sound *if for all PPT adversaries $\mathcal{A}$, we have*

$$\mathsf{Adv}^{\mathsf{sound}}_{\mathcal{A},\Pi}(\lambda) = \Pr\left[\mathsf{Expt}^{\mathsf{sound}}_{\mathcal{A},\Pi}(\lambda) = 1\right] < \mathsf{negl}(\lambda). \tag{15}$$

2) UNFORGEABILITY

Unforgeability ensures that no user can spend more than the number of tickets received or issued. Unforgeability also guarantees that the adversary address is accused whenever a ticket is double-spending. The game is formalized in Figure 2.

Definition 12 (Unforgeability): An anonymous transferable scheme is unforgeable *if for all PPT adversaries $\mathcal{A}$, we have*

$$\mathsf{Adv}^{\mathsf{Unforg}}_{\mathcal{A},\Pi}(\lambda) = \Pr\left[\mathsf{Expt}^{\mathsf{Unforg}}_{\mathcal{A},\Pi}(\lambda) = 1\right] < \mathsf{negl}(\lambda). \tag{16}$$

3) EXCULPABILITY

Exculpability ensures that an honest user can not wrongly be accused of double-spending. Exculpability also guarantees that the adversary's address is accused whenever a ticket is double-spending. Specifically, it guarantees that the adversary can not produce double-spending coins that can output

```
Experiment: $\mathsf{Expt}^{\mathsf{Unforg}}_{\mathcal{A},\Pi}(\lambda)$
1 : $\mathsf{param} \leftarrow \mathsf{ParamGen}(1^\lambda)$
2 : $(\mathsf{sk}_\mathcal{B}, \mathsf{pk}_\mathcal{B}) \leftarrow \mathsf{BKeyGen}(\mathsf{param})$
3 : $(\mathsf{id}, \pi, \mathsf{sk}_\mathcal{A}, \mathsf{pk}_\mathcal{A}) \leftarrow \mathcal{A}^{\mathtt{BRegist},\mathtt{BWith},\mathtt{BDepo}}(\mathsf{param}, \mathsf{pk}_\mathcal{B})$
4 : return 1 if there exists $(\mathsf{id}, \pi') \in \mathcal{DC}$ with $\pi \neq \pi'$
    and all of the following conditions hold:
      1) $\perp \leftarrow \mathsf{Identify}(\mathsf{id}, \pi, \pi')$
    $\wedge$ 2) $(\mathsf{pk}, \Pi_G) \leftarrow \mathsf{Identify}(\mathsf{id}, \pi, \pi')$
        $\wedge\ 0 \leftarrow \mathsf{VerifyGuilt}(\mathsf{pk}, \Pi_G)$
    $\wedge$ 3) $\mathsf{pk} \leftarrow \mathsf{Identify}(\mathsf{id}, \pi, \pi') \wedge \mathsf{pk} \notin \mathcal{UL}$
5 : return 1 if $\mathsf{id} \notin \mathcal{SC}$
      $\mathsf{Deposit}\Big(\mathcal{U}(\mathsf{id}, \pi, \mathsf{sk}_\mathcal{A}, \mathsf{pk}_\mathcal{A}, \mathsf{pk}_\mathcal{B}),$
                  $\mathcal{B}(\mathsf{sk}_\mathcal{B}, \mathsf{pk}_\mathcal{B}, \mathsf{pk}_\mathcal{A}, \mathcal{L})\Big) = \mathsf{OK}$
6 : else return 0
```

FIGURE 2. Game for *unforgeability*.

```
Experiment: $\mathsf{Expt}^{\mathtt{excul}}_{\mathcal{A},\Pi}(\lambda)$
1 : $\mathsf{param} \leftarrow \mathsf{ParamGen}(1^\lambda)$; $\mathsf{sk}_\mathcal{B} \leftarrow \mathcal{A}(\mathsf{param})$
2 : $(\mathsf{id}^*, \pi^*, \Pi^*_G) \leftarrow \mathcal{A}^{\mathtt{URegist},\mathtt{Spy},\mathtt{UWith},\mathtt{Rcv},\mathtt{S\&R},\mathtt{Depo}}(\mathsf{param})$
3 : return 1 if all of the following conditions hold:
      1) $\mathsf{VerifyGuilt}(\mathsf{id}^*, \pi^*, \Pi^*_G) = 1$
      Let $i^*$ be the owner of $(\mathsf{id}^*, \pi^*)$
    $\wedge$ 2) there was no call $\mathtt{Spy}(i^*)$
    $\wedge$ 3) $\mathsf{uds}_{i^*} = 0$
4 : else return 0
```

FIGURE 3. Game for *exculpability*.

the secret key of a user who has not committed double-spending attacks. The game is formalized in Figure 3.

Definition 13: An anonymous transferable scheme is exculpable *if for all PPT adversaries $\mathcal{A}$, we have*

$$\mathsf{Adv}^{\mathtt{excul}}_{\mathcal{A},\Pi}(\lambda) = \Pr\left[\mathsf{Expt}^{\mathtt{excul}}_{\mathcal{A},\Pi}(\lambda) = 1\right] < \mathsf{negl}(\lambda). \quad (17)$$

C. ANONYMITY NOTIONS

Anonymity notion for transferable electronic cash systems is first defined in Okamoto and Ohta [32]. Canard and Gouget [4] defined the four levels of anonymity for transferable electronic cash systems:

- *Weak anonymity* satisfies the property that any PPT adversary can not link the withdrawal and the deposit views. Still, the adversary may link two independent payments by the same user.
- *Strong anonymity* satisfies the weak anonymity, and the adversary can not link two payments by the same user. Still, the adversary may recognize the coin that he has previously observed.
- *Full anonymity* satisfies strong anonymity, and the adversary can not recognize any coin that is transferred between honest users. Still, the adversary may recognize the coin he has previously owned.
- *Perfect anonymity* satisfies full anonymity, and the adversary can not decide whether a coin is the one that he has previously owned or not.

```
Experiment: $\mathsf{Expt}^{\mathtt{c\text{-}an}}_{\mathcal{A},\Pi,b}(\lambda)$
1 : $\mathsf{param} \leftarrow \mathsf{ParamGen}(1^\lambda)$
2 : $\mathsf{pk}_\mathcal{B} \leftarrow \mathcal{A}(\mathsf{param})$
3 : $i_0^{(0)} \leftarrow \mathcal{A}^{\mathtt{URegist},\mathtt{Spy}}$; run $\mathtt{UWith}(i_0^{(0)})$ with $\mathcal{A}$
4 : $i_0^{(1)} \leftarrow \mathcal{A}^{\mathtt{URegist},\mathtt{Spy}}$; run $\mathtt{UWith}(i_0^{(1)})$ with $\mathcal{A}$
5 : $\Big((i_1^{(0)}, \cdots, i_{k_0}^{(0)}), (i_1^{(1)}, \cdots, i_{k_1}^{(1)})\Big)$
        $\leftarrow \mathcal{A}^{\mathtt{URegist},\mathtt{Spy}}$
6 : if $k_0 \neq k_1$ then return 0
7 : for $j = 1, \cdots, k_0$:
8 :     run $\mathtt{S\&R}(i_{j-1}^{(0)}, i_j^{(0)})$
9 :     run $\mathtt{S\&R}(i_{j-1}^{(1)}, i_j^{(1)})$
10 : run $\mathtt{Spd}(i_j^{(b)})$ with $\mathcal{A}$
11 : run $\mathtt{Spd}(i_j^{(1-b)})$ with $\mathcal{A}$
12 : $b^* \leftarrow \mathcal{A}$; return $b^*$
```

FIGURE 4. Game for *coin anonymity*.

According to the above anonymity notion, the scheme provided by Okamoto and Ohta [32] satisfies weak anonymity. Whereas, transferable electronic cash schemes proposed by Chaum *et al.* [33] and Canard *et al.* [34] satisfy strong anonymity. Perfect anonymity is proved to be impossible [4]. Intuitively, this proof is conducted as follows: suppose that a PPT adversary received coins c_0 and c_1 after spending a coin c such that one of the coins is related to c. Given one of the coins c_b for $b \in \{0, 1\}$, the adversary can always distinguish whether c_b is related to c or not just by depositing c_0 together with c. $b = 0$ if the adversary is accused of double-spending, $b = 1$ otherwise. Thus, the best achievable anonymity notions are full anonymity and restricted variants of perfect anonymity. More recently, Bauer *et al.* [5] introduced new notions of user anonymity, coin anonymity and coin transparency. As detailed later, all of these three notions are restricted variants of perfect anonymity by Canard and Gauget [4].

1) COIN ANONYMITY (c-an)

Definition 14 (Coin Anonymity): For all PPT adversaries $\mathcal{A}$, there exists $^\exists\lambda_0 \in \mathbb{N}$ such that for all security parameter $\lambda \geq \lambda_0$, the protocol of VeloCash*, Π_{VC} satisfies **Coin anonymity** if the following holds*:

$$\mathsf{Adv}^{\mathtt{c\text{-}an}}_{\mathcal{A},\Pi}(\lambda) = \left|\Pr\left[\mathsf{Expt}^{\mathtt{c\text{-}an}}_{\mathcal{A},\Pi,1}(\lambda) = 1\right] - \Pr\left[\mathsf{Expt}^{\mathtt{c\text{-}an}}_{\mathcal{A},\Pi,0}(\lambda) = 1\right]\right| < \mathsf{negl}(\lambda). \quad (18)$$

2) USER ANONYMITY (u-an)

We describe *User anonymity* game in Figure 5. The adversary issues or receives a ticket and sends it to one of two user

Experiment: $\mathsf{Expt}^{\texttt{u-an}}_{\mathcal{A},\Pi,b}(\lambda)$

```
1 :  $\mathsf{param} \leftarrow \mathsf{ParamGen}(1^\lambda)$
2 :  $\mathsf{pk}_\mathcal{B} \leftarrow \mathcal{A}(\mathsf{param})$
3 :  $(i_0^{(0)}, i_0^{(1)}) \leftarrow \mathcal{A}^{\texttt{URegist},\texttt{Spy}}$
4 :  run $\texttt{Rcv}(i_0^{(b)})$ with $\mathcal{A}$
5 :  $\left((i_1^{(0)},\cdots,i_{k_0}^{(0)}),(i_1^{(1)},\cdots,i_{k_1}^{(1)})\right)$
         $\leftarrow \mathcal{A}^{\texttt{URegist},\texttt{Spy}}$
6 :  if $k_0 \neq k_1$ then return 0
7 :  for $j = 1,\cdots,k_0$:
8 :    run $\texttt{S\&R}(i_{j-1}^{(b)}, i_j^{(b)})$
9 :  run $\texttt{Spd}(i_j^{(b)})$ with $\mathcal{A}$
10 : $b^* \leftarrow \mathcal{A}^{\texttt{BDepo}}$; return $b^*$
```

FIGURE 5. Game for ***user anonymity***.

groups. Then, the adversary receives the ticket again, which has been passed through between the users, and determines which of two user groups it has passed through.

Definition 15 (User Anonymity): We define User anonymity if the following properties satisfy: For all PPT adversaries $\mathcal{A}$, there exists $^\exists\lambda_0 \in \mathbb{N}$ such that for all security parameter $\lambda \geq \lambda_0$,

$$\begin{aligned}&\mathsf{Adv}^{\texttt{u-an}}_{\mathcal{A},\Pi}(\lambda)\\ &\quad = \Pr\left[\mathsf{Expt}^{\texttt{u-an}}_{\mathcal{A},\Pi,1}(\lambda)=1\right] - \Pr\left[\mathsf{Expt}^{\texttt{u-an}}_{\mathcal{A},\Pi,0}(\lambda)=1\right]\\ &\quad < \mathsf{negl}(\lambda).\end{aligned}$$

3) COIN TRANSPARENCY (c-tr)

The experiment is specified in Figure 6. We assume $\mathsf{Expt}^{\texttt{c-tr}}_{\mathcal{A},\Pi,b}(\lambda)$ aborts when challenge coins (c_0, c_1) are double-spent. For more details, see the original definition in [5]. This is a strong anonymity notion, meaning that the user who has transferred a coin will not be able to recognize it if she receives it again. The adversary issues or receives two tickets and sends them to the two user groups, respectively. Then, the adversary receives one of the tickets and determines which group the ticket has passed through.

Definition 16 (Coin Transparency): We define Coin transparency if the following properties satisfy: For all PPT adversaries $\mathcal{A}$,

$$\begin{aligned}&\mathsf{Adv}^{\texttt{c-tr}}_{\mathcal{A},\Pi}(\lambda)\\ &\quad = \left|\Pr\left[\mathsf{Expt}^{\texttt{c-tr}}_{\mathcal{A},\Pi,1}(\lambda)=1\right] - \Pr\left[\mathsf{Expt}^{\texttt{c-tr}}_{\mathcal{A},\Pi,0}(\lambda)=1\right]\right|\\ &\quad < \mathsf{negl}(\lambda). \qquad (19)\end{aligned}$$

Definition 17 (Anonymity): For $\mathsf{x} \in \{\texttt{c-an}, \texttt{u-an}, \texttt{c-tr}\}$, *an anonymous transferable scheme satisfies x if it satisfies the following equations: For all PPT adversaries $\mathcal{A}$,*

$$\begin{aligned}&\mathsf{Adv}^{\mathsf{x}}_{\mathcal{A},\Pi}(\lambda)\\ &\quad = \left|\Pr\left[\mathsf{Expt}^{\mathsf{x}}_{\mathcal{A},\Pi,1}(\lambda)=1\right] - \Pr\left[\mathsf{Expt}^{\mathsf{x}}_{\mathcal{A},\Pi,0}(\lambda)=1\right]\right|\\ &\quad < \mathsf{negl}(\lambda). \qquad (20)\end{aligned}$$

Experiment: $\mathsf{Expt}^{\texttt{c-tr}}_{\mathcal{A},\Pi,b}(\lambda)$

```
1 :  $\mathsf{param} \leftarrow \mathsf{ParamGen}(1^\lambda)$
2 :  $((\mathsf{sk}_\mathcal{W}, \mathsf{sk}_\mathcal{D}, \mathsf{sk}_\mathcal{CK}), \mathsf{pk}_\mathcal{B}) \leftarrow \mathsf{BKeyGen}(\mathsf{param})$
3 :  $i^{(0)} \leftarrow \mathcal{A}^{\texttt{URegist},\texttt{BDepo},\texttt{Spy}}(\mathsf{param}, \mathsf{pk}_\mathcal{B}, \mathsf{sk}_\mathcal{W}, \mathsf{sk}_\mathcal{D})$
4 :  $i^{(1)} \leftarrow \mathcal{A}^{\texttt{URegist},\texttt{BDepo},\texttt{Spy}}(\mathsf{param}, \mathsf{pk}_\mathcal{B}, \mathsf{sk}_\mathcal{W}, \mathsf{sk}_\mathcal{D})$
5 :  Run $\texttt{Rcv}(i^{(0)})$ with $\mathcal{A}$
6 :  Run $\texttt{Rcv}(i^{(1)})$ with $\mathcal{A}$
7 :  $\left((i_1^{(0)},\cdots,i_{k_0}^{(0)}),(i_1^{(1)},\cdots,i_{k_1}^{(1)})\right)$
         $\leftarrow \mathcal{A}^{\texttt{URegist},\texttt{Spy}}$
8 :  if $k_0 \neq k_1$ then return 0
9 :  for $j = 1,\cdots,k_0$:
10 :   run $\texttt{S\&R}(i_{j-1}^{(b)}, i_j^{(b)})$
11 : run $\texttt{Spd}(i_j^{(b)})$ with $\mathcal{A}$
12 : $b^* \leftarrow \mathcal{A}$; return $b^*$
```

FIGURE 6. Game for ***Coin transparency***.

Note that in this paper, we assume that the winning amount β is equal for all escrow accounts ϵ, because if the winning amount β is very high, the winning ticket may not satisfy the anonymity notions. Consider the case where a ticket returns to the same sender again, and the ticket is won. The user can identify the ticket from the amount spent and the number of times the ticket has been transferred.

This is a trivial assumption and is a setting to simplify the conditions for achieving anonymity. In Proportional Fee Scheme, there is a concern that the high winning amount causes the speed of payment to be slowed down since the difference between the value of the ticket and the winning amount is large. For example, if the winning amount is \$10,000 and the ticket value is 1 cent, it is not surprising that people would wait for the winning result before using it to pay. For Proportional Fee Scheme to work practically, it is ideal if the winning amount is not very high, i.e., $\beta - \gamma$ is negligibly small, where γ is the blockchain transaction fee.

D. ATTESTED EXECUTION SECURE PROCESSORS (AESP)

Tamper-proof is a property of a secure processor that prevents the leakage of sensitive information such as cryptographic keys and other confidential information against non-invasive attacks such as side-channel attacks and invasive attacks such as reverse engineering. There is no known theoretical implementation method. In many practical cases, the hardware is referred to as "tamper-proof" that has passed exhaustive penetration tests such as FIPS 140 and AVA_VAN.5 in CC certification (ISO/IEC 15408).

Several methods have been proposed in academic and commercial literature to achieve tamper-proof secure processors, As a *de facto* standard, it is often referred to as Trusted Execution Environment (TEE). TEE allows any program to be executed in secure processors. Furthermore, TEE also has the "attested execution" function of verifying that the program's output in the device is from a legitimate tamper-proof device

```
On initialize:
    // Interact with issuer $\mathcal{I}$
  1: $\langle \bot, \mathsf{sk} \rangle := \Sigma.\mathsf{Join}_{\mathcal{I},\mathcal{G}_{\text{att}}}\langle (\mathsf{gpk}, \mathsf{isk}), \mathsf{gpk} \rangle,\ T = \emptyset$
  2: initialize RT

On receive: $\texttt{getpk}()$ from some $\mathcal{P}$
  1: send gpk to $\mathcal{P}$

* * * * * Enclave operations:
On receive: $\texttt{install}(idx, \mathsf{prog})$ from some $\mathcal{P} \in \mathsf{reg}$ :
  1: if $\mathcal{P}$ is honest, assert $idx = sid$
  2: $eid \leftarrow \{0,1\}^{\lambda}$
  3: $T[eid, \mathcal{P}] := (idx, \mathsf{prog}, \mathbf{0})$
  4: send $eid$ to $\mathcal{P}$

On receive: $\texttt{resume}(eid, \mathsf{inp})$ from some $\mathcal{P} \in \mathsf{reg}$ :
  1: $(idx, \mathsf{prog}, \mathsf{mem}) := T[eid, \mathcal{P}]$
  2:     abort if $(eid, \mathcal{P}) \notin T$
  3: $(\mathsf{outp}, \mathsf{mem}) := \mathsf{prog}(\mathsf{inp}, \mathsf{mem})$
  4: $T[eid, \mathcal{P}] := (idx, \mathsf{prog}, \mathsf{mem})$
  5: $\sigma := \Sigma.\mathsf{Sign}_{\mathsf{sk}}(idx, eid, \mathsf{prog}, \mathsf{outp})$
  6: send $(\mathsf{outp}, \sigma)$ to $\mathcal{P}$

* * * * * Internal operations:
On call: $\texttt{commitment}(rid)$ from some $(eid, \cdot) \in T$ :
  1: $\mathsf{r} \leftarrow \mathsf{RT}[rid]$
  2: $(x, \texttt{com}) \leftarrow \Sigma.\mathsf{Com}_{\mathsf{sk}}(\mathsf{r})$
  3: return $(x, \texttt{com})$

On call: $\texttt{sign}(m; rid)$ from some $(eid, \cdot) \in T$ :
  1: $(idx, \mathsf{prog}, \mathsf{mem}) \leftarrow T[eid]$
  2: $\mathsf{r} \leftarrow \mathsf{RT}[rid]$
  3: $\sigma := \Sigma.\mathsf{Sign}_{\mathsf{sk}}(idx, eid, \mathsf{prog}, m; \mathsf{r})$
  4: return $(idx, eid, \mathsf{prog}, m, \sigma)$
```

FIGURE 7. The algorithms of $\mathcal{G}_{\text{att}}[\Sigma, \mathsf{reg}, \mathsf{gpk}, \mathsf{AE}]$.

by signing the output with the signing key unique to the tamper-proof device.

From the above, if $\mathcal{G}_{\text{att}}$ exists, we can achieve the followings:

1) Any arbitrary programs can be installed in $\mathcal{G}_{\text{att}}$.
2) The installed programs are obfuscated; thus, the adversary can not obtain any information more than input and output.
3) Secure channel can be achieved with the installed programs by Diffie–Hellman key exchange protocol, etc.
4) Outputs from the installed programs are signed by $\mathcal{G}_{\text{att}}$; thus, a verifier can verify that the output is sent from $\mathcal{G}_{\text{att}}$.

The signatures on outputs from the internal program by $\mathcal{G}_{\text{att}}$ should be anonymous. Since even though the programs exchange encrypted data through a secure channel, anonymity is lost if the signer's identity is known. Anonymous cryptographic technology that can not determine from which device the signature came is called "Anonymous Attestation".

Realizing anonymous attestation is achieved using a digital signature scheme called *Direct Anonymous Attestation (DAA)*, similar to group signatures. DAA differs from group signatures in that it has strong anonymity, in that even the group manager can not identify which key was used to create the signature.

E. AESP WITH DAA

Pass *et al.* have proposed formal abstractions for attested execution secure processors as *Attested Execution Secure Processors (AESP)* [6].

The structure of AESP is shown in Figure 7,8. Pass *et al.* have proposed an ideal function $\mathcal{G}_{\text{att}}$ that abstractly captures the essence of the wide range of attested execution processors. The most naive abstraction of $\mathcal{G}_{\text{att}}$ has a public key and a secret key pair $(\mathsf{gpk}, \mathsf{msk})$ for signing embedded by a manufacturer. By sharing the same secret keys among all $\mathcal{G}_{\text{att}}$, no one can distinguish the issuer of the attested signatures.

The signature on message m with the signing key is denoted as $\Sigma.\mathsf{Sign}_{\mathsf{sk}}(idx, eid, \mathsf{prog}, m; \mathsf{r})$.

$\mathcal{G}_{\text{att}}$ has four interfaces as follows:

1) Initialization:
   Generates the key pair $(\mathsf{mpk}, \mathsf{msk})$ to be used for attested execution and initializes the internal memory T.
2) Obtaining the public key:
   Outputs the public key mpk to be used for signature verification of attested execution.
3) Registration of program:
   Register a program prog in $\mathcal{G}_{\text{att}}$ and allocate unique memory space mem for the program. Enclave instance is a pair of $(idx, \mathsf{prog}, \mathsf{mem})$ for idx, which is the session ID when prog is registered. eid is the identifier of the enclave instance and is recorded in T for each platform $\mathcal{P}$.
4) Execution of program:
   Input inp to the prog specified by eid and $\mathcal{P}$, and output outp. Then, sign outp with msk and output the signature σ for the attestation.

When $\mathcal{G}_{\text{att}}$ is executed, the output outp is always signed with the embedded signing key msk. Based on the output and the signature, a verifier can verify that the output is sent from a secure processor $\mathcal{G}_{\text{att}}$.

F. EXTENSION OF AESP

We propose the following extension to the original AESP to revoke malicious platforms. We assume that AESP utilizes any direct anonymous attestation (DAA) schemes with Fiat-Shamir proof of knowledge for NP relation. The idea is the following: When a platform $\mathcal{P}$ receives a transaction, we force $\mathcal{P}$ to commit to a set of one-time randomness

```
On input: initialize() :
 1: unspentCoin := $\emptyset$
 2: txChain := $\emptyset$
 3: Key := $\emptyset$
 4: A := $\emptyset$

On input: set escrow() :
 1: $rid_\epsilon \leftarrow \mathbb{Z}$
 2: $(x, \mathtt{com}) := \mathcal{G}_{\text{att}}.\mathtt{commitment}(rid_\epsilon)$
 3: $\epsilon :=$ Hash$(x, \mathtt{com})$
 4: store unspendCoin$[\epsilon] := rid_\epsilon$
 5: return $\epsilon$

On input: init keyex$(sid, g)$ :
 1: $a \leftarrow \mathbb{Z}_p$
 2: A$[sid] := a$
 3: store A
 4: return $g^a$

On input: get addr$(sid, g^a, \sigma)$ :
 1: accept $\sigma$ if
 2:     Verify(gpk, Priv-RL, Sig-RL, $g^a, \widehat{\sigma}) = 1$
 3:    // assume up to date Priv-RL, Sig-RL is stored in prog$_w$
 4: $b \in \mathbb{Z}_p$, K $:= (g^a)^b$
 5: store Key$[sid]$ := K
 6: $rid \leftarrow \mathbb{Z}$
 7: $(x, \mathtt{com}) := \mathcal{G}_{\text{att}}.\mathtt{commitment}(rid)$
 8: addr := Hash$(x, \mathtt{com})$
 9: $\widehat{\text{addr}}$ := AE.Enc$_K$(addr)
10: store unspentCoin$[\widehat{\text{addr}}] := rid$
11: return $\widehat{\text{addr}}, (g^a, g^b)$

On input: make payment$(\widehat{\text{addr}}_X, (sid, eid, \text{prog}_w, \widehat{\text{addr}}_Y, (g^a, g^b), \sigma_Y))$ :
    // Payment from addr$_X$ to addr$_Y$
 1: accept if
 2:    Verify(gpk, Priv-RL, Sig-RL,
          $(idx, eid, \text{prog}_w, \widehat{\text{addr}}_Y, (g^a, g^b)), \sigma_Y) = 1$
 3: restore $a$
 4: store K $:= (g^a)^b$
 5: addr$_Y$ := AE.Dec$_K(\widehat{\text{addr}}_Y)$
 6: if $\widehat{\text{addr}}_X \in$ txChain
 7:    $rid_X :=$ unspentCoin$[\widehat{\text{addr}}_X]$
 8:    $(x, \mathtt{com}) := \mathcal{G}_{\text{att}}.\mathtt{commitment}(rid_X)$
 9:    addr$_X$ := Hash$(x, \mathtt{com})$
10:    restore $\tau :=$ txChain$[\widehat{\text{addr}}_X]$
11:       s.t. $\tau = \epsilon \prec \tau_1 \prec \cdots \prec \tau_n$
12:       where $\tau_n = (\text{addr}_Z \rightarrow \text{addr}_X, \text{Hash}(\tau_{n-1}), \sigma_Z)$ is
    a valid transaction to $X$ from some $Z$
13:    $\tau_{n+1} := (\text{addr}_X \rightarrow \text{addr}_Y, \text{Hash}(\tau_n))$
14:    $(sid, eid, \text{prog}_w, \tau_{n+1}, \sigma_X) := \mathcal{G}_{\text{att}}.\mathtt{sign}(\tau_{n+1}; rid_X)$
15:    delete unspentCoin$[\widehat{\text{addr}}_X]$
16:    $\widehat{\tau} :=$ AE.Enc$_K(sid, eid, \text{prog}_w, (\epsilon \prec \tau_1 \prec \cdots \prec \tau_{n+1}), \sigma_X)$
17: if $\widehat{\text{addr}}_X \notin$ txChain $\wedge$ $\widehat{\text{addr}}_X \in$ unspentCoin
18:    // this must be the case $\widehat{\text{addr}}_X = \epsilon$
19:    $rid_\epsilon :=$ unspentCoin$[\widehat{\text{addr}}_X]$
20:    $(x, \mathtt{com}) := \mathcal{G}_{\text{att}}.\mathtt{commitment}(rid_\epsilon)$
21:    $\epsilon :=$ Hash$(x, \mathtt{com})$
22:    $\tau_1 := (\epsilon \rightarrow \text{addr}_Y, \text{Hash}(\epsilon))$
23:    $(sid, eid, \text{prog}_w, (\epsilon \prec \tau_1), \sigma_X)$
          $:= \mathcal{G}_{\text{att}}.\mathtt{sign}((\epsilon \prec \tau_1); rid_\epsilon)$
24:    delete unspentCoin$[\widehat{\text{addr}}_X]$
25:       $\widehat{\tau} :=$ AE.Enc$_K(sid, eid, \text{prog}_w, (\tau \prec \tau_1), \sigma_X)$
26:    abort if $\epsilon \notin$ unspentCoin
27: return $\widehat{\tau}$

On input: receive payment$(sid, eid, \text{prog}_w, \widehat{\tau}, \widehat{\sigma}, \widehat{\text{addr}})$ :
 1: restore K := Key$[sid]$
 2: $(sid, eid, \text{prog}_w, \tau, \sigma) :=$ AE.Dec$_K(\widehat{\tau})$
    //assume $\tau = \{\epsilon \prec \cdots \tau_n\}$ for some $n \geq 1$
 3: accept if the outer and inner attested signatures are valid
    // assume up to date Priv-RL, Sig-RL and $\mathbb{C}$ are stored in
    prog$_w$
 4:    Verify(gpk, Priv-RL, Sig-RL,
 5:              $(sid, eid, \text{prog}_w, \widehat{\tau}), \widehat{\sigma}) = 1$
 6:       // for the attested signature on the encrypted ticket
 7:    Verify(gpk, Priv-RL, Sig-RL,
 8:       $(sid, eid, \text{prog}_w, \tau, \sigma) = 1$
 9:       // for the attested signature on the plaintext ticket
10:       $\epsilon \in \mathbb{C}^{\lceil k}$ for some constant $k$, and not spent
11: store txChain$[\widehat{\text{addr}}] = \tau$
12: return status$(\in \{0, 1\})$

On input: check winning$(\text{pk}_Z, \widehat{\text{addr}})$ :
 1: if $\widehat{\text{addr}} \in$ txChain
 2:    $rid :=$ unspentCoin$[\widehat{\text{addr}}]$
 3:    $(x, \mathtt{com}) := \mathcal{G}_{\text{att}}.\mathtt{commitment}(rid)$
 4:    addr := Hash$(x, \mathtt{com})$
 5:    restore $\tau_n :=$ txChain$[\widehat{\text{addr}}]$
 6: if $\tau_n \in$ win
 7:    // let $\tau_n$ has the form :
 8:    // $\tau_n = (A \rightarrow B, \text{Hash}(\tau_{\text{pre}}), \sigma)$
 9:    $\tau_{n+1} := (B \rightarrow \text{pk}_Z, \text{Hash}(\tau_{\text{pre}}))$
10: return $\tau_{n+1}$
11: if $\epsilon$ is already spent on $\mathbb{C}$, then Fork is detected
12: if txChain contains $\widehat{\text{addr}}_Z$ where $\tau_n = (\widehat{\text{addr}}_Z \rightarrow$
    $\widehat{\text{addr}}_r, \text{Hash}(\tau_{\text{pre}}), \sigma_Z)$ then Collision is detected
13: In both cases, there exists $\tau'$
14:    sk := KeyExtractor$(\tau, \tau')$
15:       return sk
16: otherwise return $\perp$
```

FIGURE 8. The algorithms of prog$_w$.

$(x, \mathtt{com})$ as the destination address of the transaction. Then, at the time when $\mathcal{P}$ transfers the transaction to other platforms, $\mathcal{P}$ must issue an anonymous attested signature committing to the randomness $(x, \mathtt{com})$. As a result, the double-spending platform must issue two different signatures using the same randomness $(x, \mathtt{com})$ so that the witness will reveal and registered in the revocation list.

In the following extension, we introduce Randomness Tape RT inside $\mathcal{G}_{\mathrm{att}}$ and give installed programs to specify randomness index rid_i.

1) RANDOMNESS TAPE

A randomness tape RT is a list of random numbers with each entry having large enough entropy.

$$\mathsf{RT} = \{rid_0, rid_1, \cdots, rid_n\} \tag{21}$$

When $\mathsf{prog_w}$ calls AESP's $\mathtt{commitment}$, $\mathcal{G}_{\mathrm{att}}$ returns $(x, \mathtt{com})$. Next, when $\mathsf{prog_w}$ calls $\mathtt{sign}$ with the same rid on $\mathtt{commitment}$, $\mathcal{G}_{\mathrm{att}}$ returns the signature σ from the random tape value corresponding to the eid.

2) INTERNAL SIGNATURE

Normally, $\mathcal{G}_{\mathrm{att}}$'s EPID signature is applied to the output outp of $\mathsf{prog_w}$ when $\mathsf{resume}(eid, \mathsf{inp})$ is called. When the ticket is sent, the ticket is encrypted with the shared key of Diffie–Hellman key exchange, and the winning ticket is registered in the blockchain, including the EPID signature on each transmission. If the signature is performed on the ciphertext, anonymity can not be satisfied because the winner will post the ciphertext and the signature on the blockchain.

We extend $\mathcal{G}_{\mathrm{att}}$ to provide internal operations that allow $\mathsf{prog_w}$ to request EPID signing on an arbitrary message to $\mathcal{G}_{\mathrm{att}}$. The $\mathsf{prog_w}$ requests $\mathcal{G}_{\mathrm{att}}$ to $\mathtt{sign}(m; rid)$ to obtain an EPID signature on m.

First, $\mathsf{prog_w}$ creates a plain-text $(\mathsf{addr}_X \rightarrow \mathsf{addr}_Y)$ indicating the transfer from X to Y, then obtains the signature as follows:

$$\sigma = \Sigma.\mathsf{Sign_{sk}}\Bigg(idx, eid, \mathsf{prog_w}, \Bigg(idx, eid, \mathsf{prog_w}, \\ (\mathsf{addr}_X \rightarrow \mathsf{addr}_Y, \mathsf{Hash}(\tau_{\mathrm{pre}}))\Bigg); \mathsf{r}\Bigg). \tag{22}$$

Then, $\mathsf{prog_w}$ encrypts the above plain-text and the signature, and $\mathcal{G}_{\mathrm{att}}$ applies an EPID signature on it and sends the following items to the payee's AESP.

$$\begin{cases} \widehat{\tau} = \mathsf{AE.Enc_K}\Bigg(idx, eid, \mathsf{prog_w}, \\ \qquad\qquad (\mathsf{addr}_X \rightarrow \mathsf{addr}_Y, \mathsf{Hash}(\tau_{\mathrm{pre}})), \sigma\Bigg) \\ \widehat{\sigma} = \sum.\mathsf{Sign_{sk}}(idx, eid, \mathsf{prog_w}, \widehat{\tau}). \end{cases} \tag{23}$$

The plain text and the EPID signature on it that will eventually appear on the blockchain will be encrypted between AESPs so that nobody can track the history of transmission from outside the AESP.

3) COMMON ID VALUES

In the original definition of AESP, at the end of the process of $\mathtt{resume}$, a signature is performed that makes it verifiable that the output is from AESP. The signature input value consists of $(idx, eid, \mathsf{prog}, \mathsf{outp})$.

In $\mathsf{VeloCash}$, we make the input values idx and eid be specified in a hash of the prog.

When the ticket is sent, idx and eid can be seen in rich environments. In the payment protocol described later, when a ticket is won, all transaction information, including past transactions, is registered in the blockchain, including the signatures. Since the winning ticket is registered in the blockchain, including idx and eid, it does not satisfy the coin transparency of the anonymity notion. By fixing idx and eid as the hash values of prog, it makes it impossible to identify from which AESP the ticket was sent.

Fixing eid means single instantiation of prog. For prog to send and receive tickets, it only needs to be able to store previously received tickets and information associated with the tickets. Since the memory corresponding to prog can store the states, single instantiation does not affect sending and receiving.

G. KEY EXTRACTOR AND REVOCATION

In $\mathsf{VeloCash}$, we would like to have a mechanism to revoke an adversary if he performs a double-spending attack. For revocation, we use the EPID revocation function.

In this paper, we do not use the EPID's signature based revocation. From the signature based revocation, the value to be registered in the revocation list is $\big(B, K(= B^f)\big)$ where f is the secret key. In $\mathsf{VeloCash}$, the signer takes a different value for B for each signature. Therefore, the signature based revocation list can not revoke the adversary.

Instead, we adopt the secret key based revocation. Once the adversary performs a double-spending attack, we make the secret key f extractable and put the f into the secret key revocation list.

We define *Key Extractor*, which is an extractor for extracting secret keys in non-interactive zero-knowledge proof using Fiat-Shamir heuristic [35], [36]. To construct the definition, we borrow the notion from Fischlin [37].

Definition 18 (Key Extractor): Suppose a Fiat-Shamir proof of knowledge for NP relation R_λ is a pair of probabilistic polynomial-time algorithms (P, V) with special soundness [37]. Then, there exists a probabilistic polynomial-time algorithm $\mathsf{KeyExtractor}$ *which holds $(x, \omega') \in R_\lambda$, for any security parameter λ satisfies the following*:

$$\begin{cases} {}^\forall(x, \omega) \in R_\lambda \textit{ and} \\ {}^\forall(x, \mathtt{com}, \mathtt{ch}, \mathtt{resp}), (x, \mathtt{com}, \mathtt{ch}', \mathtt{resp}') \\ \quad \text{s.t. } \mathsf{V}(x, \mathtt{com}, \mathtt{ch}, \mathtt{resp}) \\ \qquad\qquad = \mathsf{V}(x, \mathtt{com}, \mathtt{ch}', \mathtt{resp}') = 1 \\ \quad \textit{with } \mathtt{ch} \neq \mathtt{ch}' \\ \omega' \leftarrow \mathsf{KeyExtractor}((x, \mathtt{com}, \mathtt{ch}, \mathtt{resp}), \\ \qquad\qquad\qquad (x, \mathtt{com}, \mathtt{ch}', \mathtt{resp}')) \end{cases} \tag{24}$$

where com *is a commitment,* ch *is a challenge, and* resp *is a response.*

Lemma 2: EPID signature has the Key Extractor defined in Definition 18.

Proof: EPID signature has a form $\sigma = (B, K, T, c, s_x, s_f, s_a, s_b)$. Set the corresponding parameters in the Definition 18 as

$$(x, \texttt{com}, \texttt{ch}, \texttt{resp}) = (\{B, K, T\}, \{R_1, R_2\}, c, \{s_x, s_f, s_a, s_b\}) \quad (25)$$

where

$$\begin{aligned} R_1 &= B^{r_f} \\ R_2 &= e(T, g_2)^{-r_x} \cdot e(h_1, g_2)^{r_f} \\ &\quad \cdot e(h_2, g_2)^{r_b} \cdot e(h_2, \omega)^{r_a} \end{aligned} \quad (26)$$

and r_x, r_f, r_a, r_b are random parameters. Then, given two valid signatures $\sigma = (B, K, T, c, s_x, s_f, s_a, s_b)$, $\sigma' = (B, K, T, c', s'_x, s'_f, s'_a, s'_b)$ on the same $x = \{B, K, T\}$ and the commitment $\texttt{com} = \{R_1, R_2\}$ where

$$\begin{aligned} R_1 &= B^{r_f} = B^{s_f} K^{-c} = B^{s'_f} K^{-c'} \\ R_2 &= e(T, g_2)^{-r_x} \cdot e(h_1, g_2)^{r_f} \cdot e(h_2, g_2)^{r_b} \cdot e(h_2, \omega)^{r_a} \\ &= e(T, g_2)^{-s_x} \cdot e(h_1, g_2)^{s_f} \cdot e(h_2, g_2)^{s_b} \cdot e(h_2, \omega)^{s_a} \\ &\quad \cdot (e(g_1, g_2)/e(T, \omega))^{c} \\ &= e(T, g_2)^{-s'_x} \cdot e(h_1, g_2)^{s'_f} \cdot e(h_2, g_2)^{s'_b} \cdot e(h_2, \omega)^{s'_a} \\ &\quad \cdot (e(g_1, g_2)/e(T, \omega))^{c'} \end{aligned}$$

with $c \neq c'$, which realizes the NIZKP of the witness (x, f, a, b) by the Fiat-Shamir proof of knowledge for NP relation R_λ as follows:

$$\begin{cases} s_x = r_x + c \cdot x, & s'_x = r_x + c' \cdot x \\ s_f = r_f + c \cdot f, & s'_f = r_f + c' \cdot f \\ s_a = r_a + c \cdot a, & s'_a = r_a + c' \cdot a \\ s_b = r_b + c \cdot b, & s'_b = r_b + c' \cdot b \end{cases} \quad (27)$$

By solving the above equations, the witness (x, f, a, b) is extracted as follows:

$$\begin{aligned} &\mathsf{KeyExtractor}(\sigma, \sigma') \to (x, f, a, b) \\ &\quad = \left(\frac{s_x - s'_x}{c - c'}, \frac{s_f - s'_f}{c - c'}, \frac{s_a - s'_a}{c - c'}, \frac{s_b - s'_b}{c - c'} \right). \end{aligned} \quad (28)$$

□

Further, we construct our scheme based on the EPID signature as a representative of DAA signature schemes. However, the only DAA schemes that have the KeyExtractor are applicable to our schemes.

Theorem 5: Given two series of Fork *transactions with EPID signatures, then the double-spender's secret key can be extractable perfectly.*

Proof: Given $(\tau, \tilde{\tau}) \in$ Fork, fork detection outputs the double-spending tickets $(\tau', \tilde{\tau}')$. By inputting τ' and $\tilde{\tau}'$ into KeyExtractor, honest users can output the double-spender's secret key f.

$$\mathsf{KeyExtractor}(\tau', \tilde{\tau}') \to f. \quad (29)$$

□

H. CONSTRUCTION

This section introduces the protocol Π_{VC} of VeloCash. The general framework of the protocol is the same as Takahashi and Otsuka [1]; however, it includes a mechanism to anonymise the sending and receiving of tickets.

The protocol Π_{VC} consists of three phases: 1) Escrow Setup, 2) Payment with Lottery Ticket, and 3) Ticket Winning and Publication.

1) SETTING

The EPID's secret key based revocation list Priv-RL and signature based revocation list Sig-RL referenced among all user's AEPS shall be distributed in a blockchain and globally referenceable.

In the construction section, we have omitted the verification method of blockchain data performed by AESP. In each phase, AESP must efficiently verify that the escrow is actually registered in the blockchain and that there are no winners already by referring to the blockchain data provided by the wallet owner.

To achieve the verification, we can use the notion of *Proof of Publication* for PoW blockchains from [38], [39], which allows AESP to verify that a blockchain transaction is valid from the blockchain fragment received from the wallet owner.

In summary, it is an extension of standard transaction confirmation of the Proof-of-Work blockchain. More specifically, an AESP receives n blocks $n\text{-}T = \{B_i, \cdots, B_{i+n}\}$ and evaluates whether the time to produce the blocks is less than the specified security parameter, as follows:

$$|t_i - t_{i+n}| \leq n \cdot \delta \quad (30)$$

where t_i and t_{i+n} are time stamps extracted from blocks B_i and B_{i+n} respectively, and δ is a security parameter.

2) ParamGen(1^λ)

In our construction, we assume EPID as a DAA signature scheme with Fiat-Shamir proof of knowledge for NP relation. EPID consists of the five algorithms {Setup, Join, Sign, Verify, Revoke} as described in Appendix C. First, Issuer $\mathcal{I}$ invokes an EPID Setup protocol and generates a pair of group public/secret keys (gpk, gsk) as follows:

$$(\mathsf{gpk}, \mathsf{gsk}) \leftarrow \mathsf{Setup}(1^\lambda)$$

where $\mathsf{gpk} = (p, G_1, G_2, G_3, g_1, g_2, g_3, h_1, h_2, w = g_2^\gamma)$ and $\mathsf{gsk} = \gamma$. G_1, G_2, G_3 are groups of order p. $g_1, h_1, h_2 \in G_1$, $g_2 \in G_2$, and $g_3 \in G_3$. It outputs $\mathsf{param} = \{\mathsf{gpk}\}$.

3) $\mathsf{BKeyGen}(\mathsf{param})$

As described earlier, blockchain-based transferable e-cash systems have the blockchain $\mathbb{C}$ in place of the bank B, and $\mathbb{C}$ has no secret information. Thus, $\mathsf{BKeyGen}(\mathsf{param})$ outputs nothing such that $(\mathsf{sk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{B}}) = (\perp, \perp)$.

4) $\mathsf{UKeyGen}(\mathsf{param})$

Let $\mathcal{P}_i$ be the platform to perform $\mathsf{UKeyGen}$ and $\mathsf{param} = \{\mathsf{gpk}\}$. Note that every $\mathcal{P}_i$ is equipped with Attested Execution Secure Processor denoted by $\mathcal{G}_{\mathrm{att}i}[\mathsf{EPID}, \mathsf{reg} = \{\mathcal{P}_i\}, \mathsf{gpk}, \mathsf{AE}]$ where AE is any symmetric-key authenticated encryption algorithm. First, $\mathcal{P}_i$ initializes $\mathcal{G}_{\mathrm{att}i}$ by invoking $\mathcal{G}_{\mathrm{att}i}.\mathtt{initialize}()$ that invokes EPID $\mathsf{Join}(\mathsf{gpk})$ protocol with $\mathcal{I}$.

$$\mathsf{sk}_i = ((A, x, y), f)$$

where (A, x, y) is a BBS+ signature on f. More concretely, let p be the group order of G_1.

$$\begin{aligned} A &= (g_1 T h_2^{y''})^{\frac{1}{x+\gamma}} \\ x &\leftarrow \mathbb{Z}_p \\ y &= y' + y'' \mod p \end{aligned}$$

where $y' \leftarrow \mathbb{Z}_p$ is randomly chosen by $\mathcal{P}_i$ and $T = h_1^f \cdot h_2^{y'}$ is sent to $\mathcal{I}$. $y'' \leftarrow \mathbb{Z}_p$ is randomly chosen by $\mathcal{I}$ and the above (A, x, y) is the BBS+ signature on f.

Next, $\mathcal{P}_i$ installs a program prog_W to $\mathcal{G}_{\mathrm{att}i}$. $\mathcal{P}_i$ randomly choose a session id $sid \in \mathbb{Z}$ and sends a message $\mathtt{install}(sid, \mathsf{prog}_\mathsf{W})$ to $\mathcal{G}_{\mathrm{att}i}$ and obtains eid from $\mathcal{G}_{\mathrm{att}i}$. Then, $\mathcal{P}_i$ initializes prog_W by sending a message $\mathtt{resume}(eid, \text{“}\mathtt{initialize}\text{”})$ to $\mathcal{G}_{\mathrm{att}i}$.

It outputs $(\mathsf{sk}_i, \perp)$ as $(\mathsf{sk}_i, \mathsf{pk}_i)$.

5) $\mathsf{Withdraw}\left(\mathcal{B}\left(\mathsf{sk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{B}}, \mathsf{pk}_i\right), \mathcal{U}\left(\mathsf{sk}_i, \mathsf{pk}_i, \mathsf{pk}_{\mathcal{B}}\right)\right)$

We only take $\mathsf{param} = \{\mathsf{gpk}\}$, sk_i as inputs and neglect $(\mathsf{sk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{B}}, \mathsf{pk}_i)$ and treat them as $(\perp, \perp, \perp)$. First, $\mathcal{P}_i$ obtains an address ϵ from $\mathcal{G}_{\mathrm{att}i}$. First, prog_W randomly chooses rid to tell $\mathcal{G}_{\mathrm{att}i}$ to use the randomness on the RandomTape specified by rid.

$$\begin{aligned} (x, \mathtt{com}) &= \mathcal{G}_{\mathrm{att}}.\mathtt{commitment}(rid) \\ \mathsf{addr} &= \mathsf{Hash}(x, \mathtt{com}) \end{aligned}$$

The address of escrow account $\epsilon = \mathsf{addr}$. The message flow of the above process is complex. See Figure 9 for actual operations between $\mathcal{G}_{\mathrm{att}}$ and prog_W.

Next, $\mathcal{P}_i$ creates a transaction $(X \rightarrow \epsilon, \beta; (p, q, \mu))$ and publish it to the blockchain $\mathbb{C}$ to send winning amount to the escrow account ϵ, where X is a normal crypto address controlled by the user i, p, q, μ are the parameters for the probabilistic transferable payments. That is, p is the lottery ticket winning probability, q is the ticket transaction fee rate, μ is the fixed number used to specify the block height to calculate VDF.

It outputs $(\epsilon, \perp)$ as a ticket (id, π).

6) $\mathsf{Spend}\left(\mathcal{U}_j(\mathsf{id}, \pi, \mathsf{pk}_{\mathcal{U}_i}), \mathcal{U}_i\left(\mathsf{sk}_{\mathcal{U}_i}, \mathsf{pk}_{\mathcal{B}}\right)\right)$

We only take $\mathsf{id}, \pi, \mathsf{sk}_{\mathcal{U}_i}$ as inputs, and neglect $\mathsf{pk}_{\mathcal{U}_i}, \mathsf{pk}_{\mathcal{B}}$ and treat them as $(\perp, \perp)$. First, to establish a secure channel, $\mathcal{U}_j$ randomly choose sid and a, and sends g^a and the attested signature σ_j to $\mathcal{U}_i$ by sending a message $\mathtt{resume}(sid, eid, \text{“}\mathtt{init\ keyex}\text{”})$ to $\mathcal{G}_{\mathrm{att}j}$. $\mathcal{U}_i$ generates his temporary receiving address addr_i from the randomness identifier rid and randomly choose b, and encrypts it with the shared secret key $\mathsf{K}\left(= (g^a)^b\right)$ and obtains $\widehat{\mathsf{addr}}_i$, and sends $\widehat{\mathsf{addr}}_i$ and g^b to $\mathcal{P}_j$ by sending a message $\mathtt{resume}(eid, (\text{“}\mathtt{get\ addr}\text{”}, sid, g^a, \sigma_j))$ to $\mathcal{G}_{\mathrm{att}j}$.

$$\begin{aligned} (x, \mathtt{com}) &= \mathcal{G}_{\mathrm{att}}.\mathtt{commitment}(rid) \\ \mathsf{addr} &= \mathsf{Hash}(x, \mathtt{com}) \\ \widehat{\mathsf{addr}} &= \mathsf{AE}.\mathsf{Enc}_\mathsf{K}(\mathsf{addr}) \end{aligned}$$

Next, prog_W of $\mathcal{P}_j$ creates a transaction and calls $\mathcal{G}_{\mathrm{att}j}.\mathtt{sign}(\tau_{n+1}; rid)$ and obtains the signature σ_j on $(sid, eid, \mathsf{prog}_\mathsf{W}, \tau_{n+1})$. The transaction consists as follows:

$$\tau_{n+1} = \left(\mathsf{addr}_j \rightarrow \mathsf{addr}_i, \mathsf{Hash}(\tau_n), \sigma_{\mathcal{U}_j}\right)$$

such that

$$\epsilon \prec \tau_1 \prec \ldots \prec \tau_n \prec \tau_{n+1}.$$

Note that the rid specified here must be the same rid used to output the receiving address when receiving the ticket. $\mathcal{P}_j$ sends the encrypted address $\widehat{\mathsf{addr}}_j$ returned from $\mathtt{resume}(eid, (\text{“}\mathtt{get\ addr}\text{”}, sid, g^a, \sigma_j))$ call to his $\mathcal{G}_{\mathrm{att}j}$ when he received the ticket. Since $\widehat{\mathsf{addr}}$ and rid are recorded within prog_W, the rid used when receiving the ticket can be reused when sending.

After that, prog_W of $\mathcal{P}_j$ encrypts $(sid, eid, \mathsf{prog}_\mathsf{W}, \tau_{n+1}, \sigma)$ with the shared secret key and obtains $\widehat{\tau}_{n+1}$. Finally, $\mathcal{P}_j$ sends $\widehat{\tau}_{n+1}$ with the attested signature $\widehat{\tau}$.

It outputs $\langle \mathrm{OK}, \widehat{\tau}^* \rangle$

The message flow of the above process is complex. See Figure 10 for actual operations between $\mathcal{G}_{\mathrm{att}}$ and prog_W.

7) $\mathsf{Deposit}\left(\mathcal{U}\left(\mathsf{id}, \pi, \mathsf{sk}_{\mathcal{U}}, \mathsf{pk}_{\mathcal{U}}, \mathsf{pk}_{\mathcal{B}}\right), \mathcal{B}\left(\mathsf{sk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{U}}, \mathcal{L}\right)\right)$

We only take $\mathsf{id}, \pi, \mathsf{sk}_{\mathcal{U}}$ as inputs, and neglect $\mathsf{pk}_{\mathcal{U}}, \mathsf{sk}_{\mathcal{B}}, \mathsf{pk}_{\mathcal{B}}$ and treat them as $(\perp, \perp)$. $\mathcal{P}$ send his crypto address pk to receive the winning money and the ticket receiving address $\widehat{\mathsf{addr}}$ by sending a message

$$\mathsf{resume}(sid, eid, (\text{“}\mathtt{check\ winning}\text{”}, \mathsf{pk}, \widehat{\mathsf{addr}}))$$

to $\mathcal{G}_{\mathrm{att}}$. If the ticket satisfies the winning condition, prog_W creates a blockchain transaction $\tau := (\mathsf{addr} \rightarrow \mathsf{pk}, \mathsf{Hash}(\tau_{\mathrm{pre}}))$ and sends it with the attested signature.

It outputs $\langle \mathrm{OK}, \widehat{\tau}^* \rangle$

The message flow of the above process is complex. See Figure 11 for actual operations between $\mathcal{G}_{\mathrm{att}}$ and prog_W.

8) Identify (id, π, π')
As described earlier, blockchain-based transferable e-cash systems have the blockchain $\mathbb{C}$ in place of the bank B; this algorithm is executed by any user $\mathcal{U}$. The user receiving the ticket will verify within $\mathcal{G}_{\text{att}}$ that the ticket is a double-spending one. Detection of double-spending attacks is possible if the winning ticket is registered in the blockchain or if one user receives multiple double-spending tickets. See the Definition 9 and 10. (id, π) corresponds to $\mathsf{id} = \mathsf{Hash}(x, \mathsf{com})$ and $\pi = \sigma$, respectively. Suppose there exists double-spending tickets (id, π) and (id, π') such as

$$\mathsf{addr}_i, (\mathsf{addr}_j \to \mathsf{addr}_i, \sigma_j)$$
$$\mathsf{addr}_{i'}, (\mathsf{addr}_j \to \mathsf{addr}_{i'}, \sigma'_j)$$

where

$$\sigma_j = (B, K, T, c, s_x, s_f, s_a, s_b),$$
$$\sigma'_j = (B', K', T', c', s'_x, s'_f, s'_a, s'_b)$$

respectively. Since EPID signature has the Key Extractor from Lemma 2, the adversary's secret key is extractable from σ_j, σ'_j. The extracted secret key f will be registered in the secret key based revocation list $\mathsf{Priv\text{-}RL}$.

It outputs $(\perp, \Pi_G)$ as $(\perp, \mathsf{Priv\text{-}RL} \cup \{f\})$.

9) VerifyGuilt $((\mathsf{id}, \pi), \Pi_G)$
In blockchain-based transferable e-cash systems, this algorithm is used to verify if a ticket is created from the secret key f registered in the secret key revocation list $\mathsf{Priv\text{-}RL}$. Let

$$\mathsf{id} = (x, \mathsf{com}) = \mathsf{Hash}\left((B, K, T), (R_1, R_2)\right)$$
$$\pi = \left(B, K, T, c, s_x, s_f, s_a, s_b\right)$$

and $\Pi_G = \mathsf{Priv\text{-}RL} = \{f_1, f_2, \cdots, f_n\}$. It outputs 1 iff

$$\begin{cases} \hat{R}_1 = B^{s_f} \cdot K^{-c} \\ \hat{R}_2 = e\left(T, g_2\right)^{-s_x} \cdot e\left(h_1, g_2\right)^{s_f} \cdot e\left(h_2, g_2\right)^{s_b} \\ \qquad \cdot e\left(h_2, \omega\right)^{s_a} \cdot \left(e(g_1, g_2)/e(T, \omega)\right)^c \\ c = \mathsf{Hash}(\mathsf{gpk}, B, K, T, \hat{R}_1, \hat{R}_2, m) \\ \qquad \text{where } m = (\mathsf{addr}_j \to \mathsf{addr}_i) \\ ^{\exists}f' \in \mathsf{Priv\text{-}RL}, B^{f'} = K, \end{cases} \tag{31}$$

otherwise outputs 0.

I. TICKET TRANSFER OVERVIEW

The actual operations of the algorithms described above are processed within $\mathcal{G}_{\text{att}}$ and $\mathsf{prog_w}$, and the flow can be complex. The following describes the operations between the wallet owner (described as Wallet) and $\mathcal{G}_{\text{att}}$ and $\mathsf{prog_w}$ for setting up the escrow, making the payment, and processing the winning ticket.

Step 1: The ticket issuer issues a smart contract escrow account ϵ and confirms that ϵ has been registered in the blockchain. **Step 2:** The payee sends the ticket τ to a payee. The payee verifies that the ticket came from a legitimate wallet and that the escrow account ϵ is registered correctly in the blockchain. If there is no problem, the payee receives the ticket and returns the service or product to the payer. Then, the payee signs the ticket with his wallet and sends it to another user. **Step 3:** If the ticket received meets the requirements for lottery winning, The ticket owner can use the ticket to send the winning amount to their address.

1) ESCROW SETUP

Figure 9 shows the flow diagram. This process is part of $\mathsf{Withdraw}$. It executes a deposit transaction τ_l on the output ϵ and registers τ_l to the blockchain network B. The issuer's wallet W_X requests an escrow account ϵ from $\mathcal{G}_{\text{att}}$. After the wallet obtains ϵ from $\mathcal{G}_{\text{att}}$, W_X creates a transaction τ_l that transfer the winning amount β to ϵ and sends it to the blockchain network.

2) PAYMENT WITH LOTTERY TICKET

Figure 10 shows the flow diagram which shows sending a ticket from X to Y. Suppose that the X's wallet has a ticket τ_n and generates τ_{n+1} or generates τ_1 from the escrow account ϵ.

First, the sender X's wallet W_X resumes "`init keyex`" to perform Diffie-Hellman key exchange with the payer Y, and requests invoice to Y's wallet W_Y. $\mathcal{G}_{\text{att}}$ in W_Y generates the receiving address and encrypts it with the DH shared secret key, then sends it to W_X.

Second, W_X processes ticket transfers to the address sent by W_Y. W_X passes to $\mathcal{G}_{\text{att}}$ the address received from W_Y and the encrypted address $\widehat{\mathsf{addr}_X}$ used to receive the ticket or create an escrow account in the past.

3) TICKET WINNING AND REVOCATION

When the ticket satisfies the winning condition, the ticket owner sends the winning ticket to the blockchain network. See Figure 11.

The ticket owner sends his address pk_Y and the encrypted address used to receive the ticket to $\mathcal{G}_{\text{att}}$. Inside $\mathcal{G}_{\text{att}}$, $\mathcal{G}_{\text{att}}$ creates and returns the transaction to transfer to the address sent by the ticket owner. Then, the ticket owner receives the transaction and sends it to the blockchain network.

If the winning ticket is a double-spending one, the adversary's secret key is extracted by Fork and $\mathsf{collision}$ detection.

VI. SECURITY ANALYSIS

In this section, we analyze whether $\mathsf{VeloCash}$ satisfies economic and anonymity properties.

Finally, we analyze whether the proportional fee scheme and double-spending attacks detection methods, a requirement of the transferable micropayment scheme, can be achieved even if the anonymity is preserved.

We construct the theorems under the following assumptions. First, we assume that the tamper-proof hardware wallet and the collision-resistant hash function exist.

Assumption 1: κ-tamper proof hardware defined in Definition 23 exists.

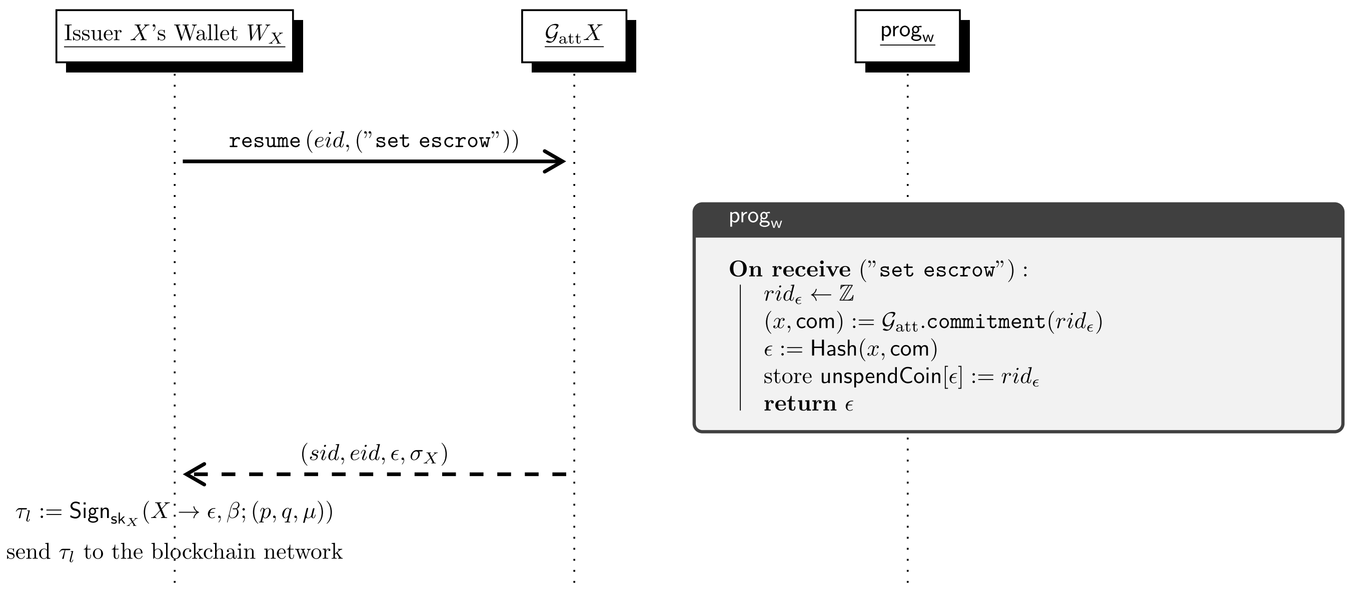

Issuer X's Wallet W_X
$\mathcal{G}_{\text{att}}X$
prog$_w$
resume $(eid, (\text{"set escrow"}))$
prog$_w$
On receive ("set escrow") :
$rid_\epsilon \leftarrow \mathbb{Z}$
$(x, \text{com}) := \mathcal{G}_{\text{att}}.\text{commitment}(rid_\epsilon)$
$\epsilon := \text{Hash}(x, \text{com})$
store unspendCoin$[\epsilon] := rid_\epsilon$
return ϵ
$(sid, eid, \epsilon, \sigma_X)$
$\tau_l := \text{Sign}_{\text{sk}_X}(X \to \epsilon, \beta; (p, q, \mu))$
send τ_l to the blockchain network

FIGURE 9. Escrow Setup.

Assumption 2: For all PPT adversaries $\mathcal{A}$ there is a negligible function negl*, collision resistant hash function* Hash *exists that satisfies the following inequality*:

$$\Pr\left[\text{Hash}(x) = \text{Hash}(x')\right] < \text{negl}(n) \qquad (32)$$

where $^\forall n \in \mathbb{N}_{>0}, x, x' \in \{0, 1\}^{|n|}$ *and* $x \neq x'$.

Next, we describe the assumptions for ensuring EPID's anonymity and unforgeability.

Assumption 3: The q-SDH problem and the Decisional Diffie-Hellman (DDH) problem are hard.

We assume that the blockchain network is agreed upon among honest users to simplify the proof.

Assumption 4: A blockchain network is agreed upon by honest users with a public parameter k satisfying the three properties the common prefix property, the chain quality property and the common-prefix property proposed by Garay et al. [29].

A. ECONOMIC PROPERTIES

In this section, we verify that the protocol of VeloCash, Π_{VC} satisfies the economic properties.

Theorem 6 (Soundness): The protocol of VeloCash, Π_{VC} *is **sound**. More formally, for all PPT adversaries $\mathcal{A}$, there exists $^\exists\lambda_0 \in \mathbb{N}$ such that for all security parameters $\lambda \geq \lambda_0$, Π_{VC} satisfies the following inequality*:

$$\mathcal{A}^{\text{sound}}_{\mathcal{A},\Pi_{VC}}(\lambda) = \Pr\left[\text{Expt}^{\text{sound}}_{\mathcal{A},\Pi_{VC}}(\lambda) = 1\right] < \text{negl}(\lambda). \quad (33)$$

Proof: Suppose there exists an AESP A_1 of wallet W_1 and another AESP A_2 of wallet W_2, and both of them are honest. The adversary wins if A_1 receives a ticket from the adversary and spends it on A_2, and A_2 refuses. Due to the tamper-proof assumption, the adversary can not break his tamper-proof wallet and can not perform double-spending attacks. Thus, to win the game, the adversary has to forge the EPID secret key and creates a ticket with a valid EPID signature. However, since the probability of forging an EPID signature is negligible small from the EPID's unforgeability in the Theorem 3, A_1 refuses to receive it. ☐

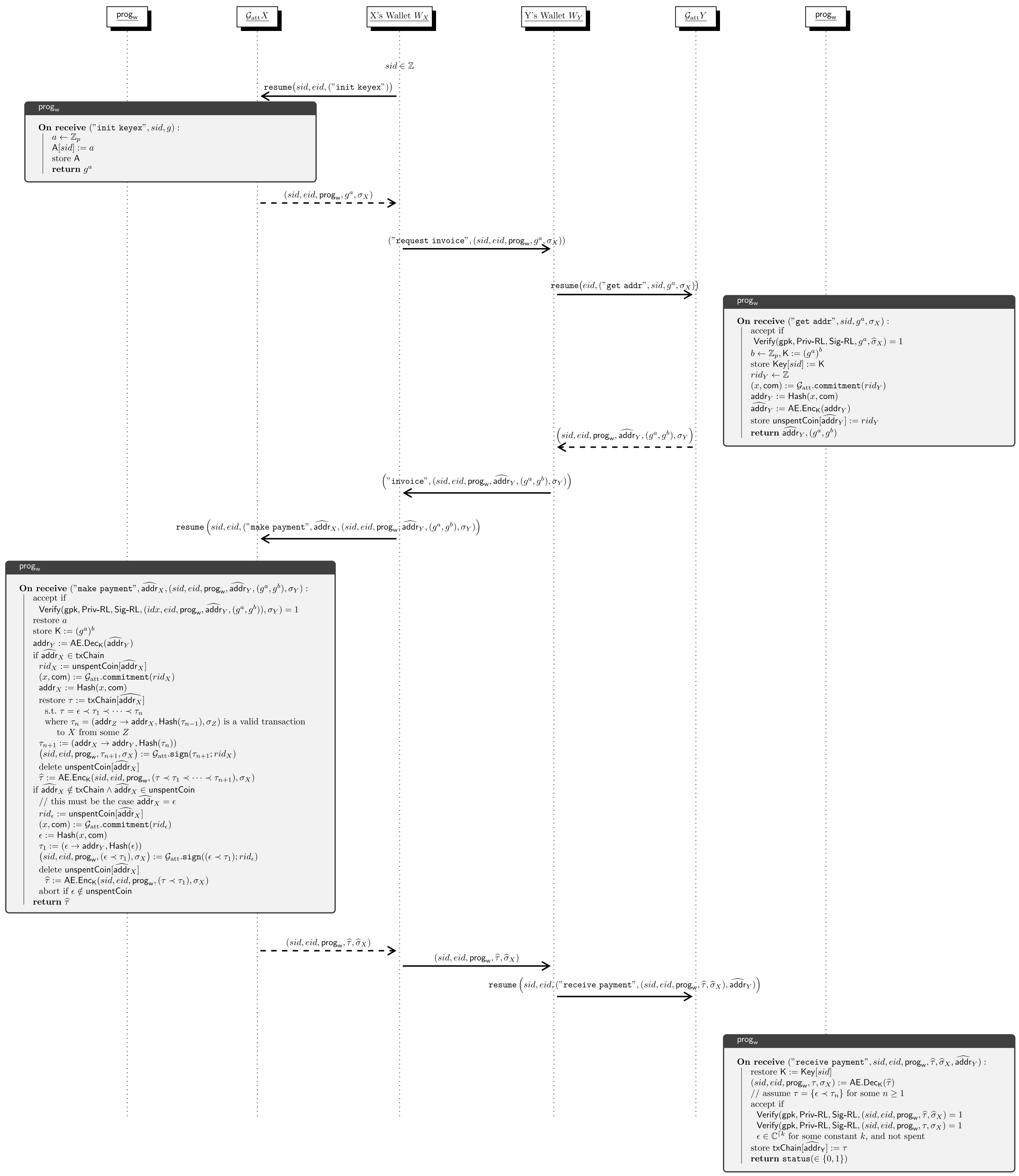
If the adversary can break his tamper-proof wallet, the adversary will perform double-spending attacks. Suppose the adversary performs double-spending attacks on A_1 and other users, and the attack is detected. In that case, this must be a case where the adversary's secret key is registered in Priv-RL, and A_2 refuses payment from A_1. Soundness is broken if $\text{sk}_\mathcal{A} \notin$ Priv-RL when A_1 receives the ticket from $\mathcal{A}$, but $\text{sk}_\mathcal{A} \in$ Priv-RL when A_2 received the ticket from A_1. However, for an adversary to gain sufficient economic benefit from a double-spending attack, a large number of double-spending would be required. From Theorem 4, there exists an upper bound on the expected utility of the attack. For users who receive double-spending tickets, it is possible to compensate for the loss by setting an appropriate issuance. See the details in [1].

The temporal collapse of soundness is a universal issue that also exists in E-Cash due to the timing gap between the execution of the attack and disabling the adversary.

Theorem 7 (Unforgeability): The protocol of VeloCash, Π_{VC} *is **unforgeable**. More formally, for all PPT adversaries $\mathcal{A}$, there exists $^\exists\lambda_0 \in \mathbb{N}$ such that for all security parameters $\lambda \geq \lambda_0$, Π_{VC} satisfies the following inequality*:

$$\mathcal{A}^{\text{Unforg}}_{\mathcal{A},\Pi_{VC}}(\lambda) = \Pr\left[\text{Expt}^{\text{Unforg}}_{\mathcal{A},\Pi_{VC}}(\lambda) = 1\right] < \text{negl}(\lambda). \quad (34)$$

Proof: The adversary wins the unforgeability game $\text{Expt}^{\text{Unforg}}_{\mathcal{A},\Pi}(\lambda)$ if he succeeds in creating a new valid ticket (id, π) which is not included in the supplied coin list $\mathcal{SC}$ or to create multiple proofs π, π' for the same ticket id, id, which is included in $\mathcal{SC}$, to get (id, π, π') but never identified


$\mathsf{prog}_{\mathsf{w}}$
$\mathcal{G}_{\mathrm{att}}X$
X's Wallet W_X
Y's Wallet W_Y
$\mathcal{G}_{\mathrm{att}}Y$
$\mathsf{prog}_{\mathsf{w}}$
$sid \in \mathbb{Z}$
$\texttt{resume}(sid, eid, (\texttt{"init keyex"}))$
$(sid, eid, \mathsf{prog}_{\mathsf{w}}, g^a, \sigma_X)$
$(\texttt{"request invoice"}, (sid, eid, \mathsf{prog}_{\mathsf{w}}, g^a, \sigma_X))$
$\texttt{resume}(eid, (\texttt{"get addr"}, sid, g^a, \sigma_X))$
$\left(sid, eid, \mathsf{prog}_{\mathsf{w}}, \widehat{\mathsf{addr}}_Y, (g^a, g^b), \sigma_Y\right)$
$\left(\texttt{"invoice"}, (sid, eid, \mathsf{prog}_{\mathsf{w}}, \widehat{\mathsf{addr}}_Y, (g^a, g^b), \sigma_Y)\right)$
$\texttt{resume}\left(sid, eid, (\texttt{"make payment"}, \widehat{\mathsf{addr}}_X, (sid, eid, \mathsf{prog}_{\mathsf{w}}, \widehat{\mathsf{addr}}_Y, (g^a, g^b), \sigma_Y)\right)$
$(sid, eid, \mathsf{prog}_{\mathsf{w}}, \widehat{\tau}, \widehat{\sigma}_X)$
$(sid, eid, \mathsf{prog}_{\mathsf{w}}, \widehat{\tau}, \widehat{\sigma}_X)$
$\texttt{resume}\left(sid, eid, (\texttt{"receive payment"}, (sid, eid, \mathsf{prog}_{\mathsf{w}}, \widehat{\tau}, \widehat{\sigma}_X), \widehat{\mathsf{addr}}_Y)\right)$

prog$_{\mathsf{w}}$

```
On receive ("init keyex", $sid$, $g$) :
    $a \leftarrow \mathbb{Z}_p$
    $\mathsf{A}[sid] := a$
    store $\mathsf{A}$
    return $g^a$
```

prog$_{\mathsf{w}}$

```
On receive ("get addr", $sid, g^a, \sigma_X$) :
    accept if
     $\mathsf{Verify}(\mathsf{gpk}, \mathsf{Priv\text{-}RL}, \mathsf{Sig\text{-}RL}, g^a, \widehat{\sigma}_X) = 1$
    $b \leftarrow \mathbb{Z}_p, \mathsf{K} := (g^a)^b$
    store $\mathsf{Key}[sid] := \mathsf{K}$
    $rid_Y \leftarrow \mathbb{Z}$
    $(x, \mathsf{com}) := \mathcal{G}_{\mathrm{att}}.\mathtt{commitment}(rid_Y)$
    $\mathsf{addr}_Y := \mathsf{Hash}(x, \mathsf{com})$
    $\widehat{\mathsf{addr}}_Y := \mathsf{AE.Enc}_{\mathsf{K}}(\mathsf{addr}_Y)$
    store $\mathsf{unspentCoin}[\widehat{\mathsf{addr}}_Y] := rid_Y$
    return $\widehat{\mathsf{addr}}_Y, (g^a, g^b)$
```

prog$_{\mathsf{w}}$

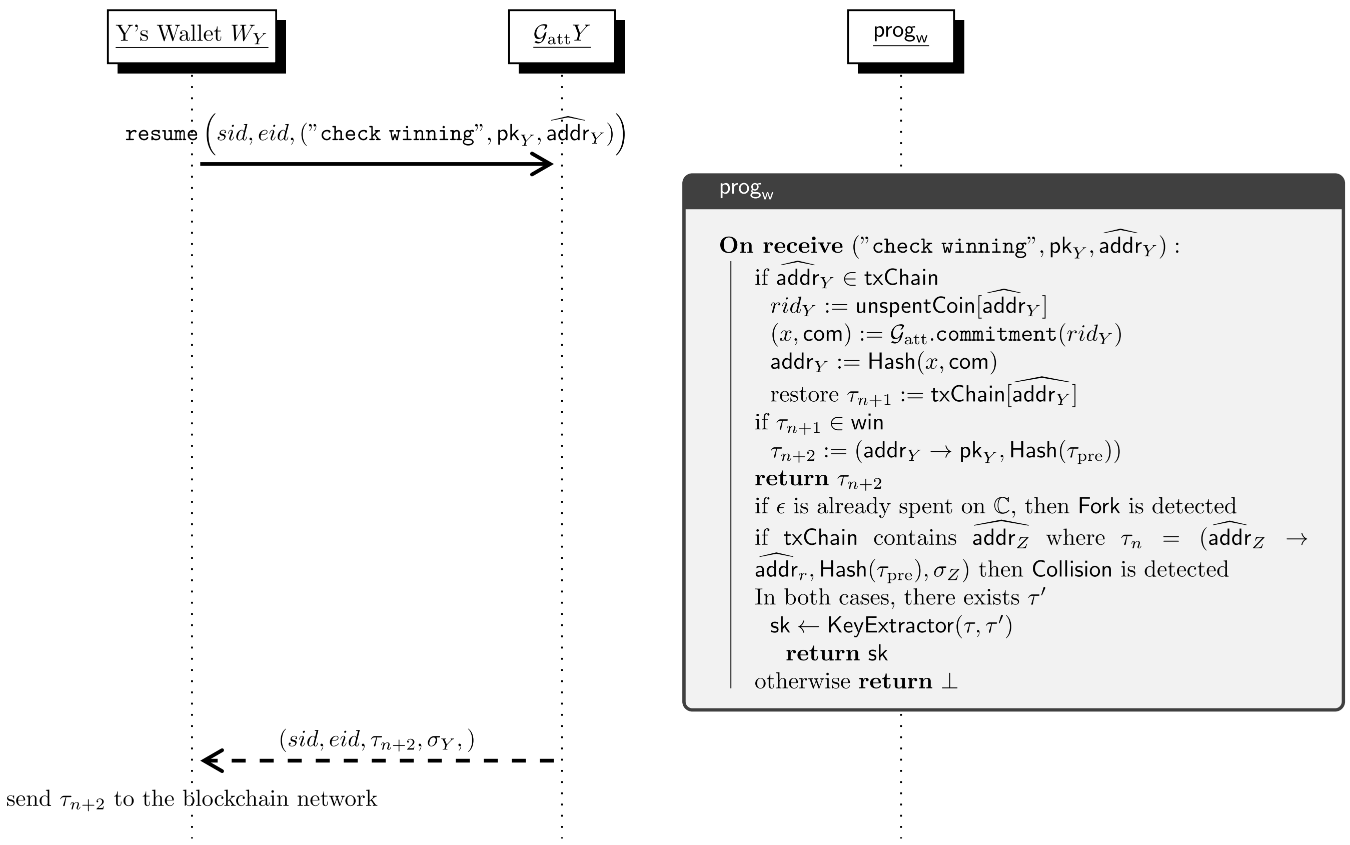
```
On receive ("make payment", $\widehat{\mathsf{addr}}_X, (sid, eid, \mathsf{prog}_{\mathsf{w}}, \widehat{\mathsf{addr}}_Y, (g^a, g^b), \sigma_Y)$ :
    accept if
      $\mathsf{Verify}(\mathsf{gpk}, \mathsf{Priv\text{-}RL}, \mathsf{Sig\text{-}RL}, (idx, eid, \mathsf{prog}_{\mathsf{w}}, \widehat{\mathsf{addr}}_Y, (g^a, g^b)), \sigma_Y) = 1$
    restore $a$
    store $\mathsf{K} := (g^a)^b$
    $\mathsf{addr}_Y := \mathsf{AE.Dec}_{\mathsf{K}}(\widehat{\mathsf{addr}}_Y)$
    if $\widehat{\mathsf{addr}}_X \in \mathsf{txChain}$
      $rid_X := \mathsf{unspentCoin}[\widehat{\mathsf{addr}}_X]$
      $(x, \mathsf{com}) := \mathcal{G}_{\mathrm{att}}.\mathtt{commitment}(rid_X)$
      $\mathsf{addr}_X := \mathsf{Hash}(x, \mathsf{com})$
      restore $\tau := \mathsf{txChain}[\widehat{\mathsf{addr}}_X]$
        s.t. $\tau = \epsilon \prec \tau_1 \prec \cdots \prec \tau_n$
        where $\tau_n = (\mathsf{addr}_Z \rightarrow \mathsf{addr}_X, \mathsf{Hash}(\tau_{n-1}), \sigma_Z)$ is a valid transaction
            to $X$ from some $Z$
      $\tau_{n+1} := (\mathsf{addr}_X \rightarrow \mathsf{addr}_Y, \mathsf{Hash}(\tau_n))$
      $(sid, eid, \mathsf{prog}_{\mathsf{w}}, \tau_{n+1}, \sigma_X) := \mathcal{G}_{\mathrm{att}}.\mathtt{sign}(\tau_{n+1}; rid_X)$
      delete $\mathsf{unspentCoin}[\widehat{\mathsf{addr}}_X]$
      $\widehat{\tau} := \mathsf{AE.Enc}_{\mathsf{K}}(sid, eid, \mathsf{prog}_{\mathsf{w}}, (\tau \prec \tau_1 \prec \cdots \prec \tau_{n+1}), \sigma_X)$
    if $\widehat{\mathsf{addr}}_X \notin \mathsf{txChain} \wedge \widehat{\mathsf{addr}}_X \in \mathsf{unspentCoin}$
      // this must be the case $\widehat{\mathsf{addr}}_X = \epsilon$
      $rid_\epsilon := \mathsf{unspentCoin}[\widehat{\mathsf{addr}}_X]$
      $(x, \mathsf{com}) := \mathcal{G}_{\mathrm{att}}.\mathtt{commitment}(rid_\epsilon)$
      $\epsilon := \mathsf{Hash}(x, \mathsf{com})$
      $\tau_1 := (\epsilon \rightarrow \mathsf{addr}_Y, \mathsf{Hash}(\epsilon))$
      $(sid, eid, \mathsf{prog}_{\mathsf{w}}, (\epsilon \prec \tau_1), \sigma_X) := \mathcal{G}_{\mathrm{att}}.\mathtt{sign}((\epsilon \prec \tau_1); rid_\epsilon)$
      delete $\mathsf{unspentCoin}[\widehat{\mathsf{addr}}_X]$
        $\widehat{\tau} := \mathsf{AE.Enc}_{\mathsf{K}}(sid, eid, \mathsf{prog}_{\mathsf{w}}, (\tau \prec \tau_1), \sigma_X)$
      abort if $\epsilon \notin \mathsf{unspentCoin}$
    return $\widehat{\tau}$
```

prog$_{\mathsf{w}}$

```
On receive ("receive payment", $sid, eid, \mathsf{prog}_{\mathsf{w}}, \widehat{\tau}, \widehat{\sigma}_X, \widehat{\mathsf{addr}}_Y$) :
    restore $\mathsf{K} := \mathsf{Key}[sid]$
    $(sid, eid, \mathsf{prog}_{\mathsf{w}}, \tau, \sigma_X) := \mathsf{AE.Dec}_{\mathsf{K}}(\widehat{\tau})$
    // assume $\tau = \{\epsilon \prec \tau_n\}$ for some $n \geq 1$
    accept if
      $\mathsf{Verify}(\mathsf{gpk}, \mathsf{Priv\text{-}RL}, \mathsf{Sig\text{-}RL}, (sid, eid, \mathsf{prog}_{\mathsf{w}}, \widehat{\tau}, \widehat{\sigma}_X) = 1$
      $\mathsf{Verify}(\mathsf{gpk}, \mathsf{Priv\text{-}RL}, \mathsf{Sig\text{-}RL}, (sid, eid, \mathsf{prog}_{\mathsf{w}}, \tau, \sigma_X) = 1$
      $\epsilon \in \mathbb{C}^{\lceil k}$ for some constant $k$, and not spent
    store $\mathsf{txChain}[\widehat{\mathsf{addr}}_{\mathsf{Y}}] := \tau$
    return $\mathtt{status}(\in \{0, 1\})$
```

FIGURE 10. **Anonymous Payment with Lottery Tickets (Payment from X to Y).**

as a double-spender by the $\mathsf{Identify}(\mathsf{id}, \pi, \pi')$ algorithm. The former case corresponds to the existential unforgeability property of EPID in VeloCash Π_{VC}. We focus on the latter case, which is further divided into three cases:



Y's Wallet W_Y
$\mathcal{G}_{\text{att}}Y$
$\mathsf{prog_w}$
$\texttt{resume}\Big(sid, eid, (\texttt{"check winning"}, \mathsf{pk}_Y, \widehat{\mathsf{addr}_Y})\Big)$
$(sid, eid, \tau_{n+2}, \sigma_Y,)$
send τ_{n+2} to the blockchain network

```
prog_w

On receive ("check winning", $\mathsf{pk}_Y$, $\widehat{\mathsf{addr}_Y}$) :
  if $\widehat{\mathsf{addr}_Y} \in \mathsf{txChain}$
    $rid_Y := \mathsf{unspentCoin}[\widehat{\mathsf{addr}_Y}]$
    $(x, \mathsf{com}) := \mathcal{G}_{\text{att}}.\texttt{commitment}(rid_Y)$
    $\mathsf{addr}_Y := \mathsf{Hash}(x, \mathsf{com})$
    restore $\tau_{n+1} := \mathsf{txChain}[\widehat{\mathsf{addr}_Y}]$
  if $\tau_{n+1} \in \mathsf{win}$
    $\tau_{n+2} := (\mathsf{addr}_Y \to \mathsf{pk}_Y, \mathsf{Hash}(\tau_{\text{pre}}))$
  return $\tau_{n+2}$
  if $\epsilon$ is already spent on $\mathbb{C}$, then Fork is detected
  if txChain contains $\widehat{\mathsf{addr}_Z}$ where $\tau_n = (\widehat{\mathsf{addr}_Z} \to \widehat{\mathsf{addr}_r}, \mathsf{Hash}(\tau_{\text{pre}}), \sigma_Z)$ then Collision is detected
  In both cases, there exists $\tau'$
    $\mathsf{sk} \leftarrow \mathsf{KeyExtractor}(\tau, \tau')$
      return sk
  otherwise return $\perp$
```

FIGURE 11. **Ticket redemption.**

Case 1: $\mathsf{Identify}(\mathsf{id}, \pi, \pi') = \perp$

This is the case that the adversary's secret key is not extractable from the double-spending tickets.

Suppose there exists

$$\begin{cases} (x, \mathtt{com}, \mathtt{ch}, \mathtt{resp}) \\ (x, \mathtt{com}, \mathtt{ch}', \mathtt{resp}') \end{cases} \tag{35}$$

such that

$$\mathsf{V}(x, \mathtt{com}, \mathtt{ch}, \mathtt{resp}) = \mathsf{V}(x, \mathtt{com}, \mathtt{ch}', \mathtt{resp}') = 1. \tag{36}$$

Since $\mathsf{KeyExtractor}$ algorithm in $\mathsf{VeloCash}$ compute the secret key f by the following equation:

$$f = \frac{s_f - s_f'}{c - c'}. \tag{37}$$

The probability that $\mathsf{KeyExtractor}$ can not output the secret key f is equal to the probability of being $\mathtt{ch} = \mathtt{ch}'$ in $(x, \mathtt{com}, \mathtt{ch}, \mathtt{resp})$ and $(x, \mathtt{com}, \mathtt{ch}', \mathtt{resp}')$. In order to be verified correctly, the two challenges $\mathtt{ch}, \mathtt{ch}'$ have to be computed, that is, c and c' in the EPID signatures have to be computed by the following equations:

$$c = \mathsf{Hash}\Big(\mathsf{gpk}, B, K, T, R_1, R_2, (\mathsf{addr}_j \to \mathsf{addr}_i, \mathsf{Hash}(\tau_{\text{pre}}))\Big)$$

$$c' = \mathsf{Hash}\Big(\mathsf{gpk}, B, K, T, R_1, R_2, (\mathsf{addr}_j \to \mathsf{addr}_{i'}, \mathsf{Hash}(\tau_{\text{pre}}))\Big). \tag{38}$$

The probability of $c = c'$ is upper bounded by the following inequality:

$$\Pr\left[c = c'\right] + \Pr\left[\mathsf{addr}_i = \mathsf{addr}_{i'}\right] < \mathsf{negl}(\lambda) \tag{39}$$

by the collision resistant property Hash functions. Note that addr_i and $\mathsf{addr}_{i'}$ are hash of randomly generated commitments (in the form of $(x, \mathtt{com})$) of honest recipients.

Case 2: $(\perp, \Pi_G) \leftarrow \mathsf{Identify}(\mathsf{id}, \pi, \pi') \wedge \mathsf{VerifyGuilt}((\mathsf{id}, \pi), \Pi_G) = 0 \wedge \mathsf{VerifyGuilt}((\mathsf{id}, \pi'), \Pi_G) = 0$
This case does not happen unless $B = 1$. Even if the adversary make $\mathsf{id} = (x, \mathtt{com}) = ((B, K, T), (R_1, R_2))$ with $B = 1$, this case is eliminated by the construction of $\mathsf{VeloCash}$ since the honest receiver does not accept the ticket as Verify algorithm fails by Equation (73).

Case 3: $\mathsf{Identify}(\mathsf{id}, \pi, \pi') = (\perp, \Pi_G \ni \mathsf{sk}) \wedge \mathsf{sk} \notin \mathcal{UL}$

This must be the case that the adversary succeeded in forging the EPID secret key f. Since the EPID's unforgeability property, the probability of succeeding is negligibly small in the security parameter λ.

Next, consider the case where the different $\mathsf{sk}'_{\mathcal{A}} (= f')$ is extracted in Equation (36), that is, the same two $\mathtt{com}$ $(= (R_1, R_2))$ are created with different secret key f. □

The following lemma states that, once $\mathsf{VerifyGuilt}$ outputs 1 on input one of the signature (id_i, π_i) with Π_G for some honest user i, then it outputs 1 for the other signatures $(\mathsf{id}_i^*, \pi_i^*)$ with Π_G of the same user i.

Lemma 3: Let $\Pi_G = \mathsf{Priv\text{-}RL}$. *Let* (id_i, π_i) *be the signatures of a user i. Then, for all honest user* $i \in \mathcal{UL}$ *and for all signature* $(\mathsf{id}_i^*, \pi_i^*)$ *generated by i, the following holds*:

$$\mathsf{VerifyGuilt}((\mathsf{id}_i, \pi_i), \Pi_G) = 1 \Rightarrow \mathsf{VerifyGuilt}((\mathsf{id}_i^*, \pi_i^*), \Pi_G) = 1 \quad (40)$$

Proof: Given the condition of the lemma, the signature (id_i, π_i) is signed by an honest user i, hence correct. From the given condition $\mathsf{VerifyGuilt}((\mathsf{id}_i, \pi_i), \Pi_G) = 1$, we see that $c = \mathsf{Hash}(\mathsf{gpk}, B, K, T, \hat{R}_1, \hat{R}_2, m)$ in Equation (31) always holds. Therefore, we focus on whether the last condition in Equation (31) holds. Let f be a part of the secret key of the user i which is used in signing the two signatures (id_i, π_i) and $(\mathsf{id}_i^*, \pi_i^*)$. We consider the following two cases:

Case 1: $f \in \Pi_G (= \mathsf{Priv\text{-}RL})$
$\mathsf{VerifyGuilt}((\mathsf{id}_i^*, \pi_i^*), \Pi_G) = 1$ always holds.

Case 2: $f \notin \Pi_G (= \mathsf{Priv\text{-}RL})$
This must be the case that B, K and B^*, K^* in σ and σ^* respectively holds the relation $B^f = K$ and $B^{*f} = K^*$ for the honest user's secret key $f \notin \Pi_G$, But, in order to satisfy the condition in Equation (40), there must exist some $f' \in \Pi_G$ different form $f \neq f'$ satisfying $B^{f'} = K$.

Recall that $B \neq 1$ is randomly chosen generator in G_3 of prime order p. Therefore,

$$B^f = K \text{ and } B^{f'} = K \Rightarrow f = f'. \quad (41)$$

This contradicts the assumption. Hence, the lemma. □

Theorem 8 (Exculpability): The protocol of ***VeloCash***, Π_{VC} *is* ***exculpable****. More formally, for all PPT adversaries* $\mathcal{A}$, *there exists* $^\exists\lambda_0 \in \mathbb{N}$ *such that for all security parameters* $\lambda \geq \lambda_0$, Π_{VC} *satisfies the following inequality*:

$$\mathsf{Adv}^{\mathtt{excul}}_{\mathcal{A},\Pi}(\lambda) = \Pr\left[\mathsf{Expt}^{\mathtt{excul}}_{\mathcal{A},\Pi}(\lambda) = 1\right] < \mathsf{negl}(\lambda). \quad (42)$$

Proof (sketch): Assume that there exists a PPT adversary $\mathcal{A}_{\mathsf{ex}}$ that can win the exculpability game and can output $(\mathsf{id}, \pi, \Pi_G)$ such that $\mathsf{VerifyGuilt}((\mathsf{id}, \pi), \Pi_G) = 1$. We show that $\mathcal{A}$ can win the EPID's anonymity game by using $\mathcal{A}_{\mathsf{ex}}$; however, this contradicts the anonymity property of EPID stated in Theorem 2. Therefore, there is no $\mathcal{A}_{\mathsf{ex}}$ that wins the exculpability game. See the detailed proof in Appendix E. □

B. ANONYMITY PROPERTIES

In this section, we verify that the protocol of $\mathsf{VeloCash}$, Π_{VC} satisfies the anonymity properties.

Assumption 5 (INT-CTXT and IND-CPA): The symmetric-key authenticated encryption scheme AE *consists of three algorithms* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *where key generation algorithm* Gen, *encryption algorithm* Enc, *and decryption algorithm* Dec. AE *is INT-CTXT and IND-CPA secure, for all PPT adversaries* $\mathcal{A}$ *there exists a negligible function* negl *such that:*

$$\mathsf{Adv}^{\mathsf{INT\text{-}CTXT}}_{\mathcal{A},\mathsf{AE}}(\lambda) = \Pr\left[\mathsf{Expt}^{\mathsf{INT\text{-}CTXT}}_{\mathcal{A},\mathsf{AE}}(\lambda) = 1\right] < \mathsf{negl}(\lambda)$$
$$\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A},\mathsf{AE}}(\lambda) = \Pr\left[\mathsf{Expt}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A},\mathsf{AE}}(\lambda) = 1\right] < \mathsf{negl}(\lambda) \quad (43)$$

where $\mathsf{Expt}^{\mathsf{INT\text{-}CTXT}}_{\mathcal{A},\mathsf{AE}}(\lambda)$ *and* $\mathsf{Expt}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A},\mathsf{AE}}(\lambda)$ *corresponds to INT-CTXT*$^{A}_{\mathcal{SE}}$ *and IND-CPA*$^{A}_{\mathcal{SE}}$ *in Bellare et al. [40], respectively.*

From Assumption 5 and Bellare *et al.* [40], we see that the symmetric-key authenticated encryption scheme AE satisfies IND-CCA secure.

Theorem 9 (IND-CCA): From Theorem 3.1 of [40], symmetric-key authenticated encryption scheme AE *is IND-CCA secure, if for all PPT adversaries* $\mathcal{A}$ *there exists a negligible function* negl *such that*:

$$\mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathcal{A},\mathsf{AE}}(\lambda) \leq 2 \cdot \mathsf{Adv}^{\mathsf{INT\text{-}CTXT}}_{\mathcal{A},\mathsf{AE}}(\lambda) + \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A},\mathsf{AE}}(\lambda) < \mathsf{negl}(\lambda) \quad (44)$$

where the probability is taken over all randomness used in the experiment.

Theorem 10 (Coin Anonymity ($\mathtt{c\text{-}an}$)*): For any* $\epsilon_0 > 0$, *the protocol of* ***VeloCash***, Π_{VC} *satisfies* ***coin anonymity*** *if* $k_0 < \frac{\epsilon_0}{2p} - 1$ *where* k_0 *is the number of challenge users per group and p is the winning probability. More formally, for all PPT adversaries* $\mathcal{A}$, *there exists* $^\exists\lambda_0 \in \mathbb{N}$ *such that for all security parameters* $\lambda \geq \lambda_0$, Π_{VC} *satisfies the following inequality*:

$$\mathsf{Adv}^{\mathtt{c\text{-}an}}_{\mathcal{A},\Pi}(\lambda) = \left|\Pr\left[\mathsf{Expt}^{\mathtt{c\text{-}an}}_{\mathcal{A},\Pi,1}(\lambda) = 1\right] - \Pr\left[\mathsf{Expt}^{\mathtt{c\text{-}an}}_{\mathcal{A},\Pi,0}(\lambda) = 1\right]\right| < \mathsf{negl}(\lambda). \quad (45)$$

Proof: Consider the case where $k_0 = 0$, that is, the case of a single challenge user. The adversary issues and spend the tickets such that

$$\widehat{\tau}_0^{(0)} = \left(\mathsf{AE.Enc}_{\mathsf{K}_0^{(0)}}\left(\epsilon^{(0)} \to i_0^{(0)}, \sigma_{\mathcal{A}}^{(0)}\right), \widehat{\sigma}_{\mathcal{A}}^{(0)}\right),$$
$$\widehat{\tau}_0^{(1)} = \left(\mathsf{AE.Enc}_{\mathsf{K}_0^{(1)}}\left(\epsilon^{(1)} \to i_0^{(1)}, \sigma_{\mathcal{A}}^{(1)}\right), \widehat{\sigma}_{\mathcal{A}}^{(1)}\right) \quad (46)$$

where $\sigma_{\mathcal{A}}^{(\cdot)}$ is the adversary's EPID signature on the message $\epsilon^{(\cdot)} \to i_0^{(\cdot)}$ and $\widehat{\sigma}_{\mathcal{A}}^{(\cdot)}$ is the adversary's EPID signature on the ciphertext $\mathsf{AE.Enc}_{\mathsf{K}_0^{(\cdot)}}(\cdot, \cdot)$.

Then, the challenge users $i_0^{(0)}$ and $i_0^{(1)}$ return the adversary the following tickets:

$$\widehat{\tau}_1^{(0)} = \left(\mathsf{AE.Enc}_{\mathsf{K}_1^{(0)}}\left(\left(\epsilon^{(0)} \to i_0^{(0)}, \sigma_{\mathcal{A}}\right), \left(i_0^{(0)} \to \mathcal{A}, \sigma_{i_0^{(0)}}\right)\right), \widehat{\sigma}_{i_0^{(0)}}\right),$$
$$\widehat{\tau}_1^{(1)} = \left(\mathsf{AE.Enc}_{\mathsf{K}_1^{(1)}}\left(\left(\epsilon^{(1)} \to i_0^{(1)}, \sigma_{\mathcal{A}}\right),\right.\right.$$

$$\left(i_0^{(1)} \to \mathcal{A}, \sigma_{i_0^{(1)}}\right)\Big), \widehat{\sigma}_{i_0^{(1)}}\Big). \tag{47}$$

The adversary can infer two partial messages as follows:

$$\begin{cases} m_0 : (\epsilon^{(0)} \to i_0^{(0)}, \cdot), (i_0^{(0)} \to \mathcal{A}, \cdot) \\ m_1 : (\epsilon^{(1)} \to i_0^{(1)}, \cdot), (i_0^{(1)} \to \mathcal{A}, \cdot) \end{cases} \tag{48}$$

such that whose signature parts are hidden from the adversary. It is easy to see that the adversary who wins the IND-CCA game with such partially hidden messages can always win the standard full message IND-CCA game. Thus, given the anonymity property of EPID signatures, we can reduce the IND-CCA game of AE to the coin anonymity game.

The above argument holds regardless of the number of challenge users $k_0 \geq 1$. This concludes the proof. □

1) OPEN PROBLEM OF COIN ANONYMITY (c-an)

If one of the challenge users wins and the ticket is registered in the blockchain, that is when $k_0 \geq \frac{\sigma_0}{2p} - 1$, the adversary can win the game in the coin anonymity game. Since the adversary issued and know the escrow account, he can see the challenge user group from the plain-text winning ticket output by $\mathsf{prog_w}$ registered in the blockchain. The reason for this is that the blockchain does not preserve anonymity. Thus, using a blockchain with anonymity, such as Confidential Transaction or ZeroCash [41] may solve the problem.

Theorem 11 (User Anonymity (u-an)): The protocol of ***VeloCash****, Π_{VC} satisfies* ***user anonymity****. More formally, for all PPT adversaries $\mathcal{A}$, there exists $^\exists\lambda_0 \in \mathbb{N}$ such that for all security parameters $\lambda \geq \lambda_0$, Π_{VC} satisfies the following inequality*:

$$\begin{aligned} &\mathsf{Adv}^{\texttt{u-an}}_{\mathcal{A},\Pi}(\lambda) \\ &\quad = \left|\Pr\left[\mathsf{Expt}^{\texttt{u-an}}_{\mathcal{A},\Pi,1}(\lambda) = 1\right] - \Pr\left[\mathsf{Expt}^{\texttt{u-an}}_{\mathcal{A},\Pi,0}(\lambda) = 1\right]\right| \\ &\quad < \mathsf{negl}(\lambda). \end{aligned}$$

Proof: Consider the case where $k_0 = 0$, that is, the case of a single challenge user. The adversary issues and spend the ticket such that

$$\widehat{\tau}_0^{(b)} = \left(\mathsf{AE.Enc}_{\mathsf{K}_0}\left(\epsilon \to i_0^{(b)}, \sigma_{\mathcal{A}}\right), \widehat{\sigma}_{\mathcal{A}}\right) \tag{49}$$

where $b \in \{0, 1\}$. Then, the challenge user returns the following tickets to the adversary.

$$\begin{aligned} \widehat{\tau}_1^{(b)} = \Bigg(\mathsf{AE.Enc}_{\mathsf{K}_1}\Bigg(\left(\left(\epsilon \to i_0^{(b)}, \sigma_{\mathcal{A}}\right)\right.&, \\ \left(i_0^{(b)} \to \mathcal{A}, \sigma_{i_0^{(b)}}\right)\Bigg)&, \widehat{\sigma}_{i_0^{(b)}}\Bigg). \end{aligned} \tag{50}$$

The adversary can infer two partial messages as follows:

$$\begin{cases} m_0 : (\epsilon^{(0)} \to i_0^{(0)}, \cdot), (i_0^{(0)} \to \mathcal{A}, \cdot) \\ m_1 : (\epsilon^{(1)} \to i_0^{(1)}, \cdot), (i_0^{(1)} \to \mathcal{A}, \cdot) \end{cases} \tag{51}$$

such that whose signature parts are hidden from the adversary. Similar to the coin anonymity game, it is easy to see that the adversary who wins the IND-CCA game with such partially hidden messages can always win the standard full message IND-CCA game. Thus, given the anonymity property of the EPID signature, we can reduce IND-CCA of AE to the user anonymity game. The above argument holds regardless of the number of challenge users $k_0 \geq 1$. This concludes the proof. □

Theorem 12 (Coin Transparency (c-tr)): The protocol of ***VeloCash****, Π_{VC} satisfies* ***coin transparency****. More formally, for all PPT adversaries $\mathcal{A}$, there exists $^\exists\lambda_0 \in \mathbb{N}$ such that for all security parameters $\lambda \geq \lambda_0$, Π_{VC} satisfies the following inequality*:

$$\begin{aligned} &\mathsf{Adv}^{\texttt{c-tr}}_{\mathcal{A},\Pi}(\lambda) \\ &\quad = \left|\Pr\left[\mathsf{Expt}^{\texttt{c-tr}}_{\mathcal{A},\Pi,1}(\lambda) = 1\right] - \Pr\left[\mathsf{Expt}^{\texttt{c-tr}}_{\mathcal{A},\Pi,0}(\lambda) = 1\right]\right| \\ &\quad < \mathsf{negl}(\lambda). \end{aligned} \tag{52}$$

Proof: The adversary receives the tickets as follows:

$$\begin{aligned} \widehat{\tau}_n^{(0)} = \Bigg(&\mathsf{AE.Enc}_{\mathsf{K}_{n-1}}\Bigg(\left(\epsilon^{(0)} \to i_1^{(0)}, \sigma_{i_0^{(0)}}\right), \\ &\left(i_1^{(0)} \to i_2^{(0)}, \sigma_{i_1^{(0)}}\right), \cdots, \\ &\left(i_{n-1}^{(0)} \to \mathcal{A}, \sigma_{i_{n-1}^{(0)}}\right)\Bigg), \widehat{\sigma}_{i_{n-1}^{(0)}}\Bigg), \\ \widehat{\tau}_n^{(1)} = \Bigg(&\mathsf{AE.Enc}_{\mathsf{K}_{n-1}}\Bigg(\left(\epsilon^{(1)} \to i_1^{(1)}, \sigma_{i_0^{(1)}}\right), \\ &\left(i_1^{(1)} \to i_2^{(1)}, \sigma_{i_1^{(1)}}\right), \cdots, \\ &\left(i_{n-1}^{(1)} \to \mathcal{A}, \sigma_{i_{n-1}^{(1)}}\right)\Bigg), \widehat{\sigma}_{i_{n-1}^{(1)}}\Bigg). \end{aligned} \tag{53}$$

Next, the adversary sends to user $i_{n+1}^{(0)}$ and $i_{n+1}^{(0)}$ the following tickets respectively.

$$\begin{aligned} \widehat{\tau}_n^{(0)} = \Bigg(&\mathsf{AE.Enc}_{\mathsf{K}_n}\Bigg(\left(\epsilon^{(0)} \to i_1^{(0)}, \sigma_{i_0^{(0)}}\right), \\ &\left(i_1^{(0)} \to i_2^{(0)}, \sigma_{i_1^{(0)}}\right), \cdots, \\ &\left(i_{n-1}^{(0)} \to \mathcal{A}, \sigma_{i_{n-1}^{(0)}}\right), \\ &\left(\mathcal{A} \to i_{n+1}^{(0)}, \sigma_{\mathcal{A}}\right)\Bigg), \widehat{\sigma}_{\mathcal{A}}\Bigg), \\ \widehat{\tau}_n^{(1)} = \Bigg(&\mathsf{AE.Enc}_{\mathsf{K}_n}\Bigg(\left(\epsilon^{(1)} \to i_1^{(1)}, \sigma_{i_0^{(1)}}\right), \\ &\left(i_1^{(1)} \to i_2^{(1)}, \sigma_{i_1^{(1)}}\right), \cdots, \\ &\left(i_{n-1}^{(1)} \to \mathcal{A}, \sigma_{i_{n-1}^{(1)}}\right), \\ &\left(\mathcal{A} \to i_{n+1}^{(1)}, \sigma_{\mathcal{A}}\right)\Bigg), \widehat{\sigma}_{\mathcal{A}}\Bigg). \end{aligned} \tag{54}$$

Then, the challenge user returns the following ticket to the adversary.

$$\begin{aligned} &\widehat{\tau}_{n+k_0+1}^{(b)} \\ &\quad = \Bigg(\mathsf{AE.Enc}_{\mathsf{K}_{n+k_0}}\Bigg(\left(\epsilon^{(b)} \to i_1^{(b)}, \sigma_{i_0^{(b)}}\right), \end{aligned}$$

$$\left(i_1^{(b)} \to i_2^{(b)}, \sigma_{i_1^{(b)}}\right), \cdots,$$
$$\left(i_{n-1}^{(b)} \to \mathcal{A}, \sigma_{i_{n-1}^{(b)}}\right),$$
$$\left(\mathcal{A} \to i_{n+1}^{(b)}, \sigma_{\mathcal{A}}\right), \cdots,$$
$$\left(i_{n+k_0}^{(b)} \to \mathcal{A}, \sigma_{i_{n+k_0}^{(b)}}\right)\Bigg)\Bigg), \widehat{\sigma}_{i_{n+k_0}^{(b)}}\Bigg) \tag{55}$$

where $b \in \{0, 1\}$. For the adversary, the n transactions until the adversary receives and signature parts of the returned ticket are hidden. Similar to the user anonymity and coin anonymity game, it is easy to see that the adversary who wins the IND-CCA game with such partially hidden messages can always win the standard full message IND-CCA game. Thus, given the anonymity property of the EPID signature, we can reduce IND-CCA of AE to the coin transparency game. This concludes the proof. □

Interestingly, in the coin transparency game, the adversary can not win the game even if the ticket is won within the challenge users. This is because the adversary does not issue the tickets and does not know the escrow accounts. Hence, the adversary has no clue to decide the group of challenge users even from the plain-text tickets registered in the blockchain.

Even if the adversary breaks the tamper-proof wallet, or if the winning ticket is registered in the blockchain and a series of transactions become known in plain text, the plain text reveals only temporary hashed values of (x, com) for each transaction and an anonymous EPID signature.

C. DOUBLE-SPENDING ATTACKS DETECTION METHODS

For double-spending attacks, the attack can be detected perfectly. Furthermore, the adversary can not profit unless the cost of destroying the hardware exceeds the maximum expected utility that he can obtain.

To achieve both Fork and Collision detection methods, two or more tickets must be given, and the series of transactions from the escrow account must be referenceable.

Definition 19 (Conditions of Detection Methods): Given two series of transactions τ and $\tilde{\tau}$. Fork and Collision Detection is achieved if and only if it satisfies the following conditions:

1) *By $\tau, \tilde{\tau}$, a series of transactions from the escrow account ϵ are referenceable.*
2) *If the attack is detected, the adversary's secret key is extracted and placed on the secret key based revocation list. Thus, the adversary will not be able to transact with honest users.*

Theorem 13 (Fork and Collision Detection): Our anonymous transfer scheme achieves Fork and Collision detection scheme perfectly.

Proof: Assume there exists forked ticket τ and $\tilde{\tau}$. Consider the fork detection. Let τ be the $\mathsf{eligible}$ ticket and registered in the blockchain, and let $\tilde{\tau}$ be the received ticket and stored in the wallet's $\mathsf{prog}_{\mathsf{W}}$. By comparing τ and $\tilde{\tau}$, the wallet can figure out the adversary's address from the forked point. Consider the Collision detection. When the wallet receives τ and $\tilde{\tau}$, it can extract the adversary's address by the fork detection and $\mathsf{KeyExtractor}$. □

Once the adversary's address is detected from the forked point, the wallet sends τ and $\tilde{\tau}$ to the EPID revocation manager $\mathcal{R}$. Then, $\mathcal{R}$ registers the adversary's secret key into the EPID's secret key based revocation list $\mathsf{Priv\text{-}RL}$.

TABLE 1. **A efficiency comparison between our VeloCash and Bauer *et al.* [5]. The coin's size is $|c_{\mathsf{bstrap}}| + k|c_{\mathsf{std}}|$ after k transfers.**

		Bauer et al. [5]	VeloCash	
Keys	$\mathsf{sk}_{\mathcal{B}}$	$9\lvert\mathbb{Z}_p\rvert + 2\lvert G_1\rvert + 2\lvert G_2\rvert$	isk	$\lvert\mathbb{Z}_p^*\rvert$
	$\mathsf{pk}_{\mathcal{B}}$	$15\lvert G_1\rvert + 8\lvert G_2\rvert$	gpk	$3\lvert G_1\rvert + 2\lvert G_2\rvert + \lvert G_3\rvert$
	$\mathsf{sk}_{\mathcal{U}}$	$\lvert\mathbb{Z}_p\rvert + 2\lvert G_1\rvert + 2\lvert G_2\rvert$	$3\lvert\mathbb{Z}_p\rvert + \lvert G_1\rvert$	
	$\mathsf{pk}_{\mathcal{U}}$	$\lvert G_1\rvert$	0	
Tickets (Coins)	Π_{guilt}	$2\lvert G_1\rvert$	$\lvert\mathbb{Z}_p\rvert$	
	c_{bstrap}	$6\lvert\mathbb{Z}_p\rvert + 147\lvert G_1\rvert + 125\lvert G_2\rvert$	$5\lvert\mathbb{Z}_p\rvert + \lvert G_1\rvert + 2\lvert G_3\rvert$	
	c_{std}	$54\lvert G_1\rvert + 50\lvert G_2\rvert$	$7\lvert\mathbb{Z}_p\rvert + \lvert G_1\rvert + 2\lvert G_3\rvert$	

VII. EFFICIENCY ANALYSIS

In this section, we present the size of each object in our proposed $\mathsf{VeloCash}$ and its comparison with Bauer *et al.* [5] in Table 1.

The setting for cyclic groups G_1, G_2 and G_3 and other elements are same as in Appendix C. Note that in Bauer et. [5] G_1, G_2 should be cyclic groups chosen that Symmetric External Diffie-Hellman assumption (SXDH) holds. On the other hand, in our $\mathsf{VeloCash}$, G_1, G_2 should be the groups that q-Strong Diffie-Hellman (q-SDH) assumption holds, and G_3 should be the group that decisional Diffie-Hellman assumption holds.

Since our $\mathsf{VeloCash}$ is a decentralized scheme and there is no online and trusted party Bank as in [5], we still need to trust to some extent, the tamper-proof device manufacturer, which is also EPID's group manager. Therefore, we compare $\mathsf{sk}_{\mathcal{B}}$ and $\mathsf{pk}_{\mathcal{B}}$ with isk and gpk, respectively.

At first glance, the ticket size (coin size) of Bauer *et al.* [5] appears much larger than of $\mathsf{VeloCash}$. This is because $\mathsf{VeloCash}$ is based on the stronger assumption (but widely used in industry), such as the existence of tamper-proof devices, whereas [5] is based purely on cryptographic assumptions.

VIII. CONCLUSION

In this paper, we have proposed $\mathsf{VeloCash}$, transferable decentralized probabilistic micropayments which preserve anonymity. For the construction of $\mathsf{VeloCash}$, we utilized a tamper-proof wallet consisting of AESP. To achieve double-spending attack detection and preserve anonymity, we add extensions to AESP that allow for randomness tape and requesting EPID signatures from prog to $\mathcal{G}_{\text{att}}$.

As discussed in Section VI, $\mathsf{VeloCash}$ satisfies the `u-an` property only for the bounded number of challenge users. This pitfall stems from the fact that blockchain accounts are not blinded. Constructing probabilistic anonymous micropayments with transferability that satisfies the anonymity notion in full is left as an open problem.

APPENDIX A
SPECIFICATION OF EPID

EPID consists of five polynomial-time algorithms:

$$\Pi_{\mathrm{EPID}} = \left(\mathsf{Setup}, \mathsf{Join}, \mathsf{Sign}, \mathsf{Com}, \mathsf{Verify}, \mathsf{Revoke}\right).$$

Setup :

$$(\mathsf{gpk}, \mathsf{isk}) \leftarrow \mathsf{Setup}(1^{\lambda}) \tag{56}$$

Issuer $\mathcal{I}$ takes the security parameter 1^{λ} as input and outputs a group public key gpk and an issuing private key isk.

Join :

$$\langle \perp, \mathsf{sk}_i \rangle \leftarrow \mathsf{Join}_{\mathcal{I},\mathcal{P}_i}\langle(\mathsf{gpk}, \mathsf{isk}), \mathsf{gpk}\rangle \tag{57}$$

Issuer $\mathcal{I}$ is given gpk and isk. $\mathcal{P}_i$ is given gpk and outputs a secret key sk_i.

Sign :

$$\sigma/\perp \leftarrow \mathsf{Sign}(\mathsf{gpk}, \mathsf{sk}, m, \mathsf{Sig\text{-}RL}) \tag{58}$$

The above is a probabilistic signature algorithm which on input gpk, sk, a message m and a signature based revocation list $\mathsf{Sig\text{-}RL}$ outputs a signature σ, or $\perp$ if sk has been revoked in $\mathsf{Sig\text{-}RL}$.
Here, we define the deterministic version of the same signature algorithm:

$$\sigma/\perp \leftarrow \mathsf{Sign}(\mathsf{gpk}, \mathsf{sk}, m, \mathsf{Sig\text{-}RL}; \mathsf{r}) \tag{59}$$

where r is a randomness. Thus, it always outputs the same signature σ if all inputs are the same.
For simplicity, we sometimes omit public parameters from expressions such that $\mathsf{Sign}_{\mathsf{sk}}(m)$ for probabilistic signature algorithms and $\mathsf{Sign}_{\mathsf{sk}}(m; \mathsf{r})$ for deterministic signature algorithms, respectively.

Com :

$$(x, \mathtt{com}) \leftarrow \mathsf{Com}(\mathsf{gpk}, \mathsf{sk}; \mathsf{r}) \tag{60}$$

The probabilistic commitment generation algorithm on input gpk, sk and r outputs x and $\mathtt{com}$. This is not defined in Brickwell and Li [7], [25]. We introduce this function for technical reasons described later in Definition 18. Assuming the signature scheme uses Fiat-Shamir proof of knowledge for NP relation $(x, \omega) \in R_{\lambda}$, there exists a key extractor, as we will see in Definition 18. When used together with the deterministic signature algorithm $\mathsf{Sign}_{\mathsf{sk}}(m; \mathsf{r})$ with the same randomness r, we can utilize these functions to identify double-spenders. Sometimes we also omit public parameters and write as $\mathtt{com}_{\mathsf{sk}}(\mathsf{r})$ for simplicity.

Verify :

$$\{0, 1\} \leftarrow \mathsf{Verify}(\mathsf{gpk}, m, \mathsf{Priv\text{-}RL}, \mathsf{Sig\text{-}RL}, \sigma)$$

On input gpk, a secret key based revocation list $\mathsf{Priv\text{-}RL}$, a signature based revocation list $\mathsf{Sig\text{-}RL}$, a message m and a signature σ, the function outputs either 1 if the signature is valid and 0 for invalid. Verify outputs 0 (invalid) when either case that σ is not a valid signature on m or that σ has been revoked.

Further, EPID [7], [25] defined the two revocation algorithms: secret key based revocation and signature based revocation, where original DAA [8] only defines the former. Our scheme does not utilize the signature based revocation, and hence our construction does not depend on the EPID signature based revocation. The two revocation algorithms are defined as follows:

Revoke - secret key based revocation

$$\mathsf{Priv\text{-}RL} \leftarrow \mathsf{Revoke}(\mathsf{gpk}, \mathsf{Priv\text{-}RL}, \mathsf{sk}) \tag{61}$$

Given gpk, $\mathsf{Priv\text{-}RL}$, and sk, $\mathcal{R}$ updates $\mathsf{Priv\text{-}RL}$ by adding sk into $\mathsf{Priv\text{-}RL}$.

Revoke - Signature based revocation

$$\mathsf{Sig\text{-}RL} \leftarrow \mathsf{Revoke}(\mathsf{gpk}, \mathsf{Priv\text{-}RL}, \mathsf{Sig\text{-}RL}, m, \sigma) \tag{62}$$

Given gpk, $\mathsf{Priv\text{-}RL}$, $\mathsf{Sig\text{-}RL}$, m, and σ, $\mathcal{R}$ updates $\mathsf{Sig\text{-}RL}$ by adding σ into $\mathsf{Sig\text{-}RL}$ after verifying σ.

APPENDIX B
SECURITY DEFINITION OF EPID

An EPID scheme is secure if it satisfies the following three requirements: correctness, anonymity, unforgeability [7], [8].

The correctness requires that every signature generated by a platform is valid except when the platform is revoked by the secret key based revocation or the signature based revocation.

Definition 20 (Correctness): An EPID scheme is correct, for every probabilistic polynomial-time adversary $\mathcal{A}$, if it satisfies the following equation:

$$\begin{cases} \sigma \leftarrow \mathsf{Sign}(\mathsf{gpk}, \mathsf{sk}, m, \mathsf{Sig\text{-}RL}), \\ \mathsf{Verify}\left(\mathsf{gpk}, m, \mathsf{Priv\text{-}RL}, \mathsf{Sig\text{-}RL}, \sigma\right) = \mathsf{valid} \end{cases}$$
$$\iff (\mathsf{sk}_i \notin \mathsf{Priv\text{-}RL}) \wedge \left(\sum\nolimits_i \cap \mathsf{Sig\text{-}RL} = \emptyset\right) \tag{63}$$

where $\sum_i$ is the set of all signatures generated by the platform $\mathcal{P}_i$.

Theorem 14 (Theorem 4 of [28]): The EPID scheme is correct.

In the anonymity game, the adversary's goal is to determine which one of two secret keys were used in generating the signature. The anonymity game between a challenger and an adversary $\mathcal{A}$ is described in Figure 12.

Definition 21 (Anonymous): An EPID scheme is anonymous, if for every probabilistic polynomial-time adversary $\mathcal{A}$, the advantage in winning the anonymity game between

```
Experiment: $\mathsf{Expt}^{\mathsf{Anon}}_{\mathcal{A}}(\lambda)$
1 : $(\mathsf{gpk}, \mathsf{isk}) \leftarrow \mathcal{A}(1^\lambda)$
2 : $(\mathtt{state}, i_0, i_1, m)$
        $\leftarrow \mathcal{A}^{\mathsf{O}_{\mathsf{Issue}}(\cdot),\mathsf{O}_{\mathsf{Join}}(\cdot),\mathsf{O}_{\mathsf{Sign}}(\mathsf{gpk},\mathsf{Sig\text{-}RL},\cdot,\cdot),\mathsf{O}_{\mathsf{Corrupt}}(\cdot)}(\mathsf{gpk})$
3 : $b \leftarrow \{0,1\}$
4 : $\sigma \leftarrow \mathsf{Sign}(\mathsf{gpk}, \mathsf{sk}_{i_b}, m, \mathsf{Sig\text{-}RL})$
5 : $b' \leftarrow \mathcal{A}^{\mathsf{O}_{\mathsf{Join}}(\cdot),\mathsf{O}_{\mathsf{Sign}}(\mathsf{gpk},\mathsf{Sig\text{-}RL},\cdot,\cdot),\mathsf{O}_{\mathsf{Corrupt}}(\cdot)}(\mathtt{state}, \sigma)$
6 : if $b = b'$, return 1
7 : else return 0
```

```
Experiment: $\mathsf{Expt}^{\mathsf{Unforg}}_{\mathcal{A}}(\lambda)$
1 : $(\mathsf{gpk}, \mathsf{isk}) \leftarrow \mathsf{Setup}(1^\lambda)$
2 : $(\mathsf{Priv\text{-}RL}^*, \mathsf{Sig\text{-}RL}^*, \sigma^*, m^*)$
        $\leftarrow \mathcal{A}^{\mathsf{O}_{\mathsf{Issue}}(\cdot),\mathsf{O}_{\mathsf{Join}}(\cdot),\mathsf{O}_{\mathsf{Sign}}(\mathsf{gpk},\mathsf{Sig\text{-}RL},\cdot,\cdot),\mathsf{O}_{\mathsf{Corrupt}}(\cdot)}(\mathsf{gpk})$
3 : return 1 if all of the following conditions hold:
        1) $\mathsf{Verify}(\mathsf{gpk}, \mathsf{Priv\text{-}RL}^*, \mathsf{Sig\text{-}RL}^*, \sigma^*, m^*) = 1$,
      $\wedge$ 2) $^{\forall} i \in U,\ \mathsf{sk}_i \in \mathsf{Priv\text{-}RL}^*$ or $^{\exists}\sigma_i \in \mathsf{Sig\text{-}RL}^*$,
      $\wedge$ 3) $\mathcal{A}$ did not obtain $\sigma^*$ by making a sign query on $m^*$
4 : else return 0
```

```
Oracle: $\mathsf{O}_{\mathsf{Join}}(i)$
1 : if $i \not\in U$
2 :     $U \leftarrow U \cup \{i\}$
3 :     generate $\mathsf{sk}_i$
4 :     return 1
5 : else return $\perp$
```

```
Oracle: $\mathsf{O}_{\mathsf{Sign}}(\mathsf{gpk}, \mathsf{Sig\text{-}RL}, i, m)$
1 : if $i \in U$
2 :     $\sigma \leftarrow \mathsf{Sign}(\mathsf{gpk}, \mathsf{sk}_i, m, \mathsf{Sig\text{-}RL})$
3 :     return $\sigma$
4 : else return $\perp$
```

```
Oracle: $\mathsf{O}_{\mathsf{Corrupt}}(i)$
1 : if $i \in U$
2 :     return $\mathsf{sk}_i$
3 : else return $\perp$
```

FIGURE 12. EPID game for ***Anonymity*** and ***Unforgeability***.

a challenger is negligible as follows:

$$\mathsf{Adv}^{\mathsf{Anon}}_{\mathcal{A}}(\lambda) = \left|2 \cdot \Pr\left[\mathsf{Expt}^{\mathsf{Anon}}_{\mathcal{A}}(\lambda) = 1\right] - 1\right| < \mathsf{negl}(\lambda) \quad (64)$$

Theorem 15 (Theorem 5 of [28]): An EPID scheme is anonymous in the random oracle model under the decisional Diffie-Hellman assumption in G_3.

Note that the adversary $\mathcal{A}$ can not make corrupt queries on the challenge users i_0 and i_1.

We say that the EPID scheme is unforgeable if no adversary can win the unforgeability game described in Figure 12. In the unforgeability game, the adversary's goal is to forge a valid signature, given that all secret keys known to the adversary have been revoked.

Definition 22 (Unforgeability): An EPID scheme is unforgeable, if for every probabilistic polynomial-time adversary $\mathcal{A}$, the advantage in winning the unforgeability game between a challenger is negligible as follows:

$$\mathsf{Adv}^{\mathsf{Unforg}}_{\mathcal{A}}(\lambda) = \Pr\left[\mathsf{Expt}^{\mathsf{Unforg}}_{\mathcal{A}}(\lambda) = 1\right] < \mathsf{negl}(\lambda) \quad (65)$$

Theorem 16 (Theorem 6 of [28]): The EPID scheme is unforgeable in the random oracle model under the strong Diffie-Hellman assumption in (G_1, G_2).

APPENDIX C CONSTRUCTION OF EPID

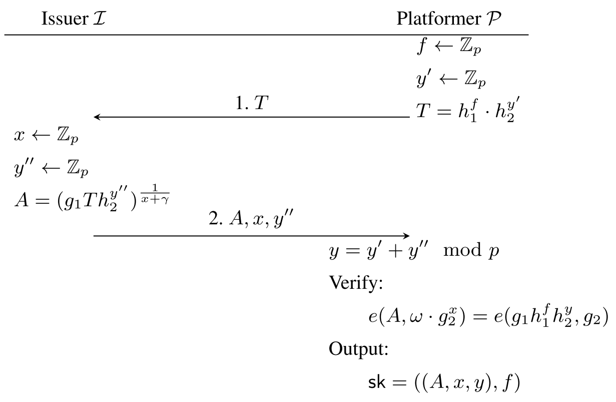
There are four types of entities in EPID: an issuer $\mathcal{I}$, a revocation manager $\mathcal{R}$, platformer $\mathcal{P}$, and verifiers $\mathcal{V}$. EPID consists of the five algorithms:

$$\Pi_{\mathsf{EPID}} = \{\mathsf{Setup}, \mathsf{Join}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Revoke}\}.$$

A. SETUP

Given 1^k, issuer $\mathcal{I}$ chooses a group pair (G_1, G_2) of prime order p and let $e : G_1 \times G_2 \to G_T$ be a bilinear map function, and a cyclic group G_3 of order p where the decisional Diffie-Hellman problem is hard. Let g_1, g_2, g_3 be the generators of G_1, G_2, G_3 respectively. $\mathcal{I}$ chooses $h_1, h_2 \leftarrow G_1$, $\gamma \leftarrow \mathbb{Z}_p^*$, and $\omega := g_2^\gamma$. This algorithm outputs

$$(\mathsf{gpk}, \mathsf{isk}) = ((p, G_1, G_2, G_3, g_1, g_2, g_3, h_1, h_2, w), \gamma).$$



Issuer $\mathcal{I}$
Platformer $\mathcal{P}$
$f \leftarrow \mathbb{Z}_p$
$y' \leftarrow \mathbb{Z}_p$
$T = h_1^f \cdot h_2^{y'}$
1. T
$x \leftarrow \mathbb{Z}_p$
$y'' \leftarrow \mathbb{Z}_p$
$A = (g_1 T h_2^{y''})^{\frac{1}{x+\gamma}}$
2. A, x, y''
$y = y' + y'' \mod p$
Verify:
$e(A, \omega \cdot g_2^x) = e(g_1 h_1^f h_2^y, g_2)$
Output:
$\mathsf{sk} = ((A, x, y), f)$

FIGURE 13. EPID Join protocol.

B. JOIN

The Join protocol is performed interactively between issuer $\mathcal{I}$ and platformer $\mathcal{P}$. The flow is described in Figure 13.

$\mathcal{I}$ has $(\mathsf{gpk}, \mathsf{isk})$ and $\mathcal{P}$ takes gpk. Finally, $\mathcal{P}$ obtains $\mathsf{sk} = ((A, x, y), f)$ where f is a unique membership key and (A, x, y) is a BBS+ signature [42] on f.

C. SIGN

Platformer $\mathcal{P}$ inputs $\mathsf{gpk}, \mathsf{sk} := ((A, x, y), f), m \in \{0,1\}^*$, and a signature based revocation list $\mathsf{Sig\text{-}RL}$, then outputs the signature σ as follows:

1) Chooses $B \leftarrow G_3$ such that $B \neq 1$ and computes

$$K := B^f \quad (66)$$

2) Chooses $a \leftarrow \mathbb{Z}_p$ and computes

$$b := y + ax, \;\; T := A \cdot h_2^a \tag{67}$$

3) It randomly picks

$$r_x \leftarrow \mathbb{Z}_p, \;\; r_f \leftarrow \mathbb{Z}_p, \;\; r_a \leftarrow \mathbb{Z}_p, \;\; r_b \leftarrow \mathbb{Z}_p \tag{68}$$

4) Computes

$$\begin{aligned} R_1 &:= B^{r_f} \\ R_2 &:= e\left(T, g_2\right)^{-r_x} \cdot e\left(h_1, g_2\right)^{r_f} \\ &\quad \cdot e\left(h_2, g_2\right)^{r_b} \cdot e\left(h_2, \omega\right)^{r_a} \end{aligned} \tag{69}$$

5) Computes

$$c = \mathsf{Hash}(\mathsf{gpk}, B, K, T, R_1, R_2, m) \tag{70}$$

6) Computes

$$\begin{aligned} s_x &= r_x + c \cdot x, \;\; s_f = r_f + c \cdot f, \\ s_a &= r_a + c \cdot a, \;\; s_b = r_b + c \cdot b \end{aligned} \tag{71}$$

7) Sets $\sigma_0 := (B, K, T, c, s_x, s_f, s_a, s_b)$

Then, $\mathcal{P}$ verifies the above output values satisfy the following signature of knowledge protocol as follows:

$$\begin{aligned} \mathrm{SPK}\big\{&(x, f, a, b) : B^f = K \wedge \\ &e\left(T, g_2\right)^{-x} \cdot e\left(h_1, g_2\right)^f \cdot e\left(h_2, g_2\right)^b \cdot e\left(h_2, w\right)^a \\ &= e(T, \omega)/e(g_1, g_2)\big\}(m) \end{aligned} \tag{72}$$

D. VERIFY

Verifier $\mathcal{V}$ inputs gpk, m, a secret key based revocation list $\mathsf{Priv\text{-}RL}$, a signature-key based revocation list $\mathsf{Sig\text{-}RL}$, and a signature σ, then verifies the signature as follows:

1) Let $\sigma = (\sigma_0, \sigma_1, \ldots, \sigma_{n_2})$, where $\sigma_0 = (B, K, T, c, s_x, s_f, s_a, s_b)$
2) Verifies

$$\begin{aligned} &B, K \overset{?}{\in} G_3 \text{ and } B \neq 1, \\ &T \overset{?}{\in} G_1, \;\; s_x, s_f, s_a, s_b \overset{?}{\in} \mathbb{Z}_p \end{aligned} \tag{73}$$

3) Computes

$$\begin{aligned} \hat{R}_1 &= B^{s_f} \cdot K^{-c} \\ \hat{R}_2 &= e\left(T, g_2\right)^{-s_x} \cdot e\left(h_1, g_2\right)^{s_f} \cdot e\left(h_2, g_2\right)^{s_b} \\ &\quad \cdot e\left(h_2, \omega\right)^{s_a} \cdot \left(e(g_1, g_2)/e(T, \omega)\right)^c \end{aligned} \tag{74}$$

4) Verifies

$$c \overset{?}{=} \mathsf{H}\left(\mathsf{gpk}, B, K, T, \hat{R}_1, \hat{R}_2, m\right) \tag{75}$$

5) Let $\mathsf{Priv\text{-}RL} = \left\{f_1, \ldots, f_{n_1}\right\}$. For $i = 1, \ldots, n_1$, it checks that $K \overset{?}{\neq} B^{f_i}$
6) Let $\mathsf{Sig\text{-}RL} = \left\{(B_1, K_1), \ldots, (B_{n_2}, K_{n_2})\right\}$. For $i = 1, \ldots, n_2$, it verifies that σ_i is a valid zero-knowledge proof,

$$\mathrm{SPK}\left\{(f) : K = B^f \wedge K_i \neq B_i^f\right\}(m). \tag{76}$$

E. REVOKE

1) SECRET KEY BASED REVOCATION

Given gpk, $\mathsf{Priv\text{-}RL}$, and $\mathsf{sk} = (A, x, y, f)$ to be revoked, revocation manager $\mathcal{R}$ updates $\mathsf{Priv\text{-}RL}$ as follows:

1) verify the correctness of sk,

$$e(A, g_2^x \omega) = e(g_1 h_1^f h_2^y, g_2) \tag{77}$$

2) appends f in σ_0 to $\mathsf{Sig\text{-}RL}$

2) SIGNATURE BASED REVOCATION

Given gpk, $\mathsf{Priv\text{-}RL}$, $\mathsf{Sig\text{-}RL}$, m, and σ to be revoked, revocation manager $\mathcal{R}$ updates $\mathsf{Sig\text{-}RL}$ as follows:

1) checks

$$\mathsf{Verify}(\mathsf{gpk}, m, \mathsf{Priv\text{-}RL}, \emptyset, \sigma_0) = \mathsf{valid} \tag{78}$$

2) appends (B, K) in σ_0 to $\mathsf{Sig\text{-}RL}$

APPENDIX D
DEFINITIONS OF PROBABILISTIC TRANSFER SCHEME

1) TAMPER-PROOF WALLET

In VeloCash, our premise is that all users participating in the transferable scheme have tamper-proof hardware wallets. The wallet consists of AESP (tamper-proof device) manufactured by a trusted manufacturer. The obfuscated program that sends and receives tickets is installed in the wallet's AESP. It does not accept any unauthorized operations that deviate from the protocol, such as double-spending tickets.

For each tamper-proof wallet, the $\mathcal{G}_{\text{att}}$ in the wallet has an EPID secret key manufactured and embedded by a trusted manufacturer. The role of EPID in VeloCash is to make it possible to verify that the ticket has been sent from a legitimate and genuine wallet.

It also plays an essential role in excluding adversaries who have performed double-spending attacks. When the adversary performs the double-spending attacks, honest users will detect the attack and put the adversary's secret key f into the secret key based revocation list $\mathsf{Priv\text{-}RL}$. This will prevent the adversaries from sending and receiving the tickets with honest users; thus, the adversary will not be able to perform double-spending attacks again.

Theoretically, a tamper-proof device is not destructive; however, the adversary can break it realistically.

Definition 23 (κ-Tamper Proof): A device is called κ-tamper proof if it satisfies the following conditions:

1) *Tamper-proof hardware is the hardware that prevents an adversary from stealing and changing stored data.*
2) *The device is either completely broken/tampered with or working perfectly with probability κ and $(1-\kappa)$, respectively.*[4]
3) *Broken or tampered is a state in which all confidential information inside the device, including the secret key, has been leaked to the adversary.*

[4]In reality, the adversaries are biased, but we assume it can not be distinguishable from a legitimate user from outside.

We assume each device is in a state either completely broken/tampered with or working perfectly. They occur with probabilities κ and $(1-\kappa)$, respectively. As long as the behaviour is observed from the outside, it is not possible to distinguish between a device operating correctly and a device that adversaries control the correct device that has access to its internal key.

2) STRUCTURE OF THE TICKET

This section describes the structure of the lottery ticket and the escrow account. The general framework is the same as [1], but with anonymity.

We denote by σ_A a signature signed by A.

Definition 24: A lottery ticket τ consists of a four-tuple:

$$\left(A, B, \tau_{pre}, \sigma\right) \tag{79}$$

where A and B are accounts of a sender and a receiver, respectively. τ_{pre} is a reference to the previous ticket or the escrow account ϵ. σ is the EPID digital signature on τ.

For readability, we write a ticket τ as:

$$\tau = \left(A \rightarrow B, \tau_{\text{pre}}\right)_X . \tag{80}$$

Let A and B both be the receiving addresses. Also, X is the signature of the sender A, which indicates the signer's identity.

We define $|\tau|$ the "number of generations" of τ, which is the length of the sequence from ϵ to τ. For example, $|\tau| = n$ if there exists a sequence $\tau_1, \ldots, \tau_{n-1}$ such that $\tau_\epsilon \prec \tau_1 \prec \tau_2 \prec \cdots \prec \tau_{n-1} \prec \tau$. We define $|\tau| = \infty$ if no such sequence exists.[5] To write compactly, we denote by τ_i the i-th generation of τ.

Definition 25 (Transferred Transaction): Two tickets $\tau_i = \left(A \rightarrow B, \tau_{pre}\right)_X$ and $\tau_{i+1} = \left(A' \rightarrow B', \tau'_{pre}\right)_{X'}$ are said to be transferred *if and only if following properties satisfies:*

$$\begin{cases} \mathsf{Hash}(\tau_i) = \tau'_{pre} \\ A = X,\ B = A' = X' \\ \sigma \textit{ is valid} \end{cases} \tag{81}$$

Then, we write $\tau_i \prec \tau_{i+1}$.

We write $\tau_i \lll \tau_{i+n}$ if there exists a sequence of ordered lottery tickets $\tau'_1 \prec \ldots \prec \tau'_n$ for $n \geq 1$ and they satisfy $\tau_i \prec \tau'_1$ and $\tau'_n \prec \tau_{i+n}$. If τ has no previous lottery tickets, the ticket is called a "genesis" ticket. For the genesis tickets τ_1 tied to an escrow account ϵ, we specially denote by $\epsilon \prec \tau_1$, so that lottery tickets are simply written as:

$$\epsilon \prec \tau_1 \prec \tau_2 \prec \ldots \prec \tau_n. \tag{82}$$

Definition 26: A lottery ticket τ is said to be valid *with respect to a blockchain $\mathbb{C}$ for some security parameter k if and only if there exists an escrow account ϵ and a sequence of transactions $\{\tau_1, \ldots, \tau_n\}$ such that*

$$\epsilon \in \mathbb{C}^{\lceil k} \quad \textit{and} \quad \epsilon \prec \tau_1 \prec \ldots \prec \tau_n \prec \tau. \tag{83}$$

[5]For practical purposes, we assume that the height of τ can only be measured when all tickets in the sequence from τ_ϵ to τ are given. Even if such a sequence exists, the height of τ is considered to be ∞ unless the entire sequence is specifically presented.

$\mathbb{C}^{\lceil k}$ denotes the set of blocks that are k or more blocks before the beginning of the blockchain. This notion is borrowed from Garay *et al.* [29].

In order to specify the parameters of the transferable transaction, the escrow account ϵ further contains:

$$(\beta, p, q, \mu) \tag{84}$$

where β is the ticket winning amount, p is the probability for determining winning a ticket, q is the lottery ticket transaction rate of proportional fee scheme ,[6] and μ is a fixed value used to determine the winning ticket.

We define an escrow as *active* when the ticket generated from the escrow is in the process of being distributed, but the ticket does not meet the condition for winning and is not registered in the blockchain.

Definition 27 (Active Escrow): We define escrow account ϵ to be active *if and only if the following conditions are met:*

$$\begin{cases} \tau \quad \textit{is } \mathsf{valid} \\ \epsilon \in \mathbb{C}^{\lceil k} \wedge \tau \notin \mathbb{C}^{\lceil k} \end{cases} \tag{85}$$

where τ is the sequence of transactions such as $\{\epsilon \prec \tau_1 \prec \cdots \prec \tau_n\}$.

3) TICKET WINNING CONDITION

This section describes the design of the ticket winnings.

Definition 28: τ_v is said to be win *if and only if the following properties satisfy:*

$$\mathsf{win} = \left\{\tau_v \;\middle|\; \mathrm{VDF}\left(h_0 + v \cdot \mu\right) < D, v \in \mathbb{N}\right\} \tag{86}$$

where v is the number of generations of τ, h_0 is the block height containing the escrow account and μ is the fixed number specified in the escrow account ϵ.

The probability p is calculated using a simple Verifiable Delay Function (VDF) [30]. The calculation can be done after a certain period of time has elapsed from when the ticket is transferred according to the number of generations. For example, if a ticket with $h_0 = 100$, $\mu = 5$ and $v = 3$ is received, the VDF value will be known when the block height of 115 is confirmed.

If the ticket $\tau \in \mathsf{win}$ has already been sent and the ownership has been transferred, the user with the latest ownership is considered eligible and can get the winning amount β.

Definition 29: τ_v is said to be eligible *if and only if the following properties satisfies:*

$$\mathsf{eligible} = \left\{\tau_v \;\middle|\; {}^{\exists}\tau_{v'} \in \mathsf{win} \wedge \tau_{v'} \lll \tau_v\right\} \tag{87}$$

eligible ticket will be considered as the final winning ticket. Thus, the user who has the eligible ticket can get β from the escrow account ϵ.

4) PROPORTIONAL FEE SCHEME

Proportional Fee Scheme is a payment scheme first introduced by Takahashi and Otsuka [1]. This scheme is where each time a payer transfers a ticket; the payer bears the fee based on the number of generations of the ticket.

[6]See Appendix D-4.

If the ticket is transferable, eventually, a transaction will be registered on the blockchain to pay the winning amount to the winner. When a very small amount of money is deposited and transferred as a micropayment, there is no benefit for the user to bear the blockchain transaction fee for the winning amount. Therefore, the winning ticket will not be claimed and in circulation forever. With the proportional fee scheme, if the winning amount exceeds the blockchain fee, there will be a motivation to claim, and the number of times a ticket can be circulated will be capped.

Definition 30 (Proportional Fee Scheme): Let q be the lottery ticket transaction fee rate. Suppose a payer sends a ticket τ_i, and the payee gives goods or services worth $(1-q)^i\beta$ to the sender. The fee borne by each payment is $(1-q)^{i-1}q\beta$.

Specifically, the fee for each payment is $\tau_{i-1} - \tau_i = (1-q)^{i-1}q\beta$, and the profit (income − expenditure) when $\tau_i = \mathsf{eligible}$ is $\beta - \left(\tau_i + \gamma\right) = \left(1-(1-q)^i\right)\beta - \gamma$ where γ is the blockchain transaction fee.

Suppose the ticket satisfies the win condition before the accumulated fees exceed the blockchain transaction fee γ. In this case, the user may decide whether to send it to the blockchain network and get β or transfer the ticket to another user as payment. Specifically, the user can profit from the eligible ticket by getting the winning amount β under the following condition:

$$\left(1-(1-q)^i\right)\beta > \gamma. \tag{88}$$

If the ticket satisfies the win condition and is transferred to another user, the ticket is distributed as eligible and can be sent to the blockchain in any subsequent generation. Naturally, the ticket will be sent to the blockchain network in the generation that satisfies the equation 88.

This scheme has the advantage that the payment fee can be smaller than the blockchain transaction fee. The average transaction fee for cryptocurrencies, especially Bitcoin, is approximate \$2 [31]. By contrast, in our transferable scheme, let $\beta = \$100$, $p = \frac{1}{100}$ and $q = \frac{1}{10}$, the fee for a \$1 transfer is about 10 cents.

Both the existing Lottery scheme and our Transferable scheme can aggregate blockchain transactions by the winning probability p. The difference is that our transferable scheme does not increase the gambling potential, even making the winning probability p smaller. In the existing scheme, the smaller p is, the lower the probability that the payee will win the ticket, making the income more unstable for the payees. In our transferable scheme, even if the ticket is not winning, the payee can use it for payment by paying a smaller fee than the blockchain transaction fee.

To achieve Proportional Fee Scheme, several conditions need to be met.

Definition 31 (Condition of Proportional Fee Scheme): In transferable payment, Proportional Fee Scheme is achieved if and only if the ticket's series of transactions from ϵ is referenceable.

There is a concern that the sizeable winning amount β decreases the velocity of the ticket. Since if there is a large gap between the winning amount β and the value of the ticket, the profit of winning $\beta - \gamma$ will be more significant. Therefore, recipients should decide whether to use the ticket for payment after confirming their winnings, which causes the velocity of the ticket to be slow. The solution is not to make the winning amount β too high. In addition, if we set the winning amount β to a value almost equal to the blockchain transaction fee γ, the profit of winning will be minimal; thus, the velocity of the ticket will not be affected.

A. DOUBLE-SPENDING ATTACKS DETECTION METHODS

We adopt tamper-proof devices for sending and receiving tickets and assume that double-spending attacks can not be performed under normal circumstances. However, in reality, the tamper-proof device can be broken, and the adversary can perform double-spending attacks. We propose two detection methods. One is *Fork Detection* which detects the attacks perfectly, and the other one is *Collision Detection* which takes places an upper bound on the expected value by the attack. When both Fork and Collision detection detect an attack, honest users share the adversary's address and revoke it so that the adversary can never participate in the payment system again.

With the two methods, we force an adversary to compare the cost of breaking the tamper-proof wallet with the expected utility obtained. In order for the adversary to profit from the attack, he has to benefit from a single attack more than the cost of breaking the tamper-proof wallet.

1) FORK DETECTION

Fork detection is a detection method that detects double-spending attacks perfectly. *Fork* implies that two tickets are assumed to exist, and the resulting ticket prefixes are identical except for k elements from both ends of the ticket.

Definition 32 (Fork of Transferred Transactions): Given two series of transactions initiated with the same escrow account $\tau = \{\epsilon \prec \tau_1 \prec \cdots \prec \tau_n\}$, and $\tilde{\tau} = \{\tilde{\epsilon} \prec \tilde{\tau}_1 \prec \cdots \prec \tilde{\tau}_{n'}\}$, the series of transactions are said to be Fork if and only if it satisfies: $^\exists k \in \mathbb{Z}_{\geq 0}$,

$$\begin{cases} \epsilon = \tilde{\epsilon} \\ \tau^{\lceil k} \lll \tilde{\tau} \wedge \tilde{\tau}^{\lceil k} \lll \tau \\ \tau^{\lceil k-1} \not\lll \tilde{\tau} \vee \tilde{\tau}^{\lceil k-1} \not\lll \tau. \end{cases} \tag{89}$$

Lemma 4 (Fork Detection): Given two series of transactions $(\tau, \tilde{\tau}) \in$ Fork, there exists an efficient fork detection algorithm that outputs the double-spending transactions.

Proof: Put the two transactions $\tau, \tilde{\tau}$ as $\tau = \{\epsilon \prec \tau_1 \prec \cdots \prec \tau_n\}$, and $\tilde{\tau} = \{\tilde{\epsilon} \prec \tilde{\tau}_1 \prec \cdots \prec \tilde{\tau}_{n'}\}$. From the Definition 9, there exists $k \geq 0$ that satisfies the condition (89). We assume $n \geq n'$ without loss of generality. Then, $\tau^* = \{\epsilon \prec \tau_1 \prec \cdots \prec \tau_{n-k}\}$ is the longest common prefix, and τ_{n-k+1} and $\tilde{\tau}_{n-k+1}$ are the double spending transactions. □

Every time one of the Fork transactions is published on the blockchain, players holding the other Fork transactions can identify the double-spending transaction perfectly using the above-described procedure. The adversary's address will be revealed as the common spender of the two resulting transactions in Lemma 1.

2) COLLISION DETECTION

The following *collision detection* upper-bounds the adversary's expected utility.

Definition 30 (Collision of Transferred Transactions): Suppose each user has multiple addresses. It is said to be a collision if and only if there exists a user who receives two transactions τ and $\tilde{\tau}$ such that $(\tau, \tilde{\tau}) \in \mathsf{Fork}$ in any of his addresses.

In the following analysis, we assume the round-based synchronous network model, where every message submitted by the users and the adversary is queued in the input tape of the recipients. The order of the messages in the queue is arbitrary permuted by the adversary. These queued messages cannot be read out until the next round starts. In practice, the time period for one round is set so that a broadcast message reaches every honest user in the network within the time period. Further, to work the collision detection effectively, we assume that the ticket transaction protocol is conducted in the three rounds: Round 1) The adversary sends the tickets to an honest payee. Round 2) The payee checks received tickets for the collisions. If the payee finds *collision*, he broadcasts the tickets, say τ and $\tilde{\tau}$, in the collision (or publishes them on the blockchain). Round 3) If the collision is not detected, the payee gives products or services to the payer in return. However, if the collision is detected, the adversary's address is rejected and will never be accepted by all honest users.

Theorem 17 (Collision Detection): Let u be the number of users who participate in the transfer scheme, where each user has $\alpha\,(\geq\ 2)$ addresses. By collision detection in the round-based synchronous network, the expected utility of double-spending attack $\mathbb{E}_d$ is upper-bounded by the following inequality:

$$\mathbb{E}_d \leq \sqrt{\frac{u}{e}}\,\beta. \tag{90}$$

Proof: As stated in the Definition 10, we assume a uniform distribution where each user has α addresses.[7] This must be the case where the adversary chooses uniformly l different addresses from the total of $\alpha \cdot u$ addresses. By the round scheme, the adversary can not profit if a single user address is chosen more than once.

Let $p(l; u)$ be the probability that at least one user address is chosen more than once. This probability is described as follows:

$$p(l; u) \approx 1 - e^{-\frac{l^2}{2u}}. \tag{91}$$

[7]In reality, the number of addresses each user has is considered more likely to follow an exponential distribution. Therefore, it is an unfavourable assumption that all users have the same number of addresses α.

Assume that the adversary double-spending l tickets with a maximum value of β per ticket. The adversary's expected utility value is

$$\mathbb{E}_d < \max_{l}\left\{ l\beta \cdot (1 - p(l; u)) \right\}. \tag{92}$$

Thus, $\mathbb{E}_d$ is at most $\sqrt{\frac{u}{e}}\beta$ when $l = \sqrt{u}$. □

In our transferable scheme, a double-spending attack is perfectly detected, and the address used in the attack will be rejected by all users. Therefore, it is not profitable for the adversary unless the cost of breaking a single tamper-proof wallet exceeds the maximum expected value gained by the attack. Specifically, the adversary can not profit under the following conditions:

$$\sqrt{\frac{u}{e}}\beta < \Phi \tag{93}$$

where Φ is the cost of breaking κ-tamper proof wallet.

As an example, consider the maximum expected utility value $\mathbb{E}_d$ with $u = 1,000,000$ and $\beta = \$100$. Applying the equation (90) produces $\mathbb{E}_d \lessapprox \$60,700$.

APPENDIX E
PROOF OF THEOREM 8

To prove exculpability property stated in Theorem 8, we assume there exists a PPT adversary $\mathcal{A}_{\mathsf{ex}}$ that wins the exculpability game defined in Definition 13 with non-negligible probability, namely, that outputs $(\mathsf{id}, \pi, \Pi_G)$ such that $\mathsf{VerifyGuilt}((\mathsf{id}, \pi), \Pi_G) = 1$.

We will show a reduction that we can construct a PPT adversary $\mathcal{A}$ which wins the EPID anonymity game defined in Definition 21 with non-negligible probability provided oracle access to $\mathcal{A}_{\mathsf{ex}}$ is available.

In order to share the same user list between $\mathcal{A}$ and $\mathcal{A}_{\mathsf{ex}}$, we will not allow $\mathcal{A}_{\mathsf{ex}}$ access to Join; instead, $\mathcal{A}$ provide a sufficiently long list of users $(i_1, \ldots, i_n)$ created by $\mathcal{A}$ and provide the list of users to $\mathcal{A}_{\mathsf{ex}}$. Note that this modification to the exculpability game does not affect the success probability of $\mathcal{A}_{\mathsf{ex}}$.

In the reduction, $\mathcal{A}$ generates the parameter param and $\mathsf{sk}_{\mathcal{B}}$, and it joins n users and creates the list of users $(i_1, \ldots, i_n)$. Then, $\mathcal{A}_{\mathsf{ex}}$ is executed with input $(\mathsf{param}, \mathsf{sk}_{\mathcal{B}}, (i_1, \ldots, i_n))$, and it outputs $(\mathsf{id}, \pi, \Pi_G)$ with non-negligible probability as follows:

$$\begin{aligned}
&\mathsf{param} \leftarrow \mathsf{ParamGen}(1^{\lambda});\ \mathsf{sk}_{\mathcal{B}} \leftarrow \mathcal{A}(\mathsf{param})\\
&(i_1, \cdots, i_n) \leftarrow \mathcal{A}^{\mathsf{O_{Join}}(\cdot)}\\
&(\mathsf{id}, \pi, \Pi_G) \leftarrow \mathcal{A}_{\mathsf{ex}}^{\mathtt{Spy,UWith,Rcv,S\&R,Depo}}\\
&\qquad (\mathsf{param}, \mathsf{sk}_{\mathcal{B}}, (i_1, \ldots, i_n))\\
&\qquad \text{such that } \mathsf{VerifyGuilt}((\mathsf{id}, \pi), \Pi_G) = 1
\end{aligned}$$

Note that the accused double-spender i^* is contained in the user list $(i_1, \ldots, i_n)$. Otherwise, given $\mathsf{VerifyGuilt}((\mathsf{id}, \pi), \Pi_G) = 1$, $\mathcal{A}_{\mathsf{ex}}$ must win the unforgeability game in Figure 2, and it contradicts with Theorem 7.

Then, we can easily find out the user i^* who created the signature (id, π) by iteratively ask all honest users to produce signatures for a message in a form of randomly chosen commitment, namely, $m = (x, \mathsf{com})$. That is, for $i^* \in \{i_1, \cdots, i_n\}$, $\mathcal{A}$ queries to $\mathsf{O_{Sign}}(\mathsf{gpk}, \mathsf{sk}_{i^*}, m, \mathsf{Sig\text{-}RL})$ and gets σ_{i_j}. By Lemma 3, there must exist $i^* \in (i_1, \ldots, i_n)$ such that $\mathsf{VerifyGuilt}((m, \sigma_{i^*}), \Pi_G) = 1$. Then, $\mathcal{A}$ randomly chooses $\bar{i^*} \in \{i_1, \ldots, i_n\} \setminus \{i^*\}$. $\mathcal{A}$ outputs $(\Pi_G, i^*, \bar{i^*}, m)$.

At step 5 in the Anonymity Game defined in Figure 12, $\mathcal{A}$ outputs $b' = 0$ if $\mathsf{VerifyGuilt}((m, \sigma), \Pi_G) = 1$ otherwise $b' = 1$.

By Lemma 3, the adversary $\mathcal{A}$ wins the Anonymity Game with non-negligible probability if there exists $\mathcal{A}_{\mathsf{ex}}$ which wins the Excupability Game defined in Figure 3 with non-negligible probability. This contradicts the anonymity property of EPID stated in Theorem 2, and this reduction is tight. Therefore, there is no $\mathcal{A}_{\mathsf{ex}}$ that wins the exculpability game with non-negligible probability. □

ACKNOWLEDGMENT

An earlier version of this paper was presented in part at the Proceedings of 2021 26th European Symposium on Research in Computer Security [DOI: 10.1007/978-3-030-88418-5_19].

REFERENCES

[1] T. Takahashi and A. Otsuka, "Probabilistic micropayments with transferability," in *Proc. Eur. Symp. Res. Comput. Secur.*, in Lecture Notes in Computer Science. Cham, Switzerland: Springer, 2021.

[2] L. Gudgeon, P. Moreno-Sanchez, S. Roos, P. McCorry, and A. Gervais, "SoK: Layer-two blockchain protocols," in *Financial Cryptography and Data Security* (Lecture Notes in Computer Science). New York, NY, USA: Springer, 2020, pp. 201–226.

[3] G. Almashaqbeh, A. Bishop, and J. Cappos, "MicroCash: Practical concurrent processing of micropayments," in *Financial Cryptography and Data Security—FC* (Lecture Notes in Computer Science). New York, NY, USA: Springer, 2020, pp. 227–244.

[4] S. Canard and A. Gouget, "Anonymity in transferable E-cash," in *Applied Cryptography and Network Security* (Lecture Notes in Computer Science), vol. 5037. Berlin, Germany: Springer, 2008, pp. 207–223, [Online]. Available: http://link.springer.com/10.1007/978-3-540-68914-0_13

[5] B. Bauer, G. Fuchsbauer, and C. Qian, "Transferable E-cash: A cleaner model and the first practical instantiation," in *Public-Key Cryptography—PKC* (Lecture Notes in Computer Science). New York, NY, USA: Springer, 2021.

[6] R. Pass, E. Shi, and F. Tramèr, "Formal abstractions for attested execution secure processors," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science). New York, NY, USA: Springer, 2017.

[7] E. Brickell and J. Li, "Enhanced privacy ID from bilinear pairing for hardware authentication and attestation," in *Proc. IEEE 2nd Int. Conf. Social Comput.*, Aug. 2010, pp. 1–8.

[8] E. Brickell, J. Camenisch, and L. Chen, "Direct anonymous attestation," in *Proc. 11th ACM Conf. Comput. Commun. Secur.* New York, NY, USA: Association for Computing Machinery, Oct. 2004, pp. 132–145.

[9] A. Miller, I. Bentov, S. Bakshi, R. Kumaresan, and P. McCorry, "Sprites and state channels: Payment networks that go faster than lightning," in *Financial Cryptography and Data Security* (Lecture Notes in Computer Science). New York, NY, USA: Springer, 2019, pp. 508–526.

[10] D. Wheeler, "Transactions using bets," in *Security Protocols* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 1997, pp. 89–92.

[11] R. L. Rivest, "Electronic lottery tickets as micropayments," in *Financial Cryptography* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 1997, pp. 307–314.

[12] R. Pass and A. Shelat, "Micropayments for decentralized currencies," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.* New York, NY, USA: Association for Computing Machinery, 2015, pp. 207–218, doi: 10.1145/2810103.2813713.

[13] A. Chiesa, M. Green, J. Liu, P. Miao, I. Miers, and P. Mishra, "Decentralized anonymous micropayments," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science). Cham, Switzerland: Springer, 2017, pp. 609–642.

[14] D. Zhang, J. Le, N. Mu, and X. Liao, "An anonymous off-blockchain micropayments scheme for cryptocurrencies in the real world," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 50, no. 1, pp. 32–42, Jan. 2020.

[15] C. Ferretti, A. Leporati, L. Mariot, and L. Nizzardo, "Transferable anonymous payments via TumbleBit in permissioned blockchains," in *Proc. 2nd Distrib. Ledger Technol. Workshop* in CEUR Workshop Proceedings, vol. 2334. Pisa, Italy: DLT@ITASEC, Feb. 2019, pp. 56–67. [Online]. Available: https://CEUR-WS.org

[16] F. Baldimtsi, M. Chase, G. Fuchsbauer, and M. Kohlweiss, "Anonymous transferable E-cash," in *Proc. IACR Int. Workshop Public Key Cryptogr.* Berlin, Germany: Springer, 2015, pp. 101–124.

[17] M. Bellare, D. Micciancio, and B. Warinschi, "Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions," in *Proc. 22nd Int. Conf. Theory Appl. Cryptograph. Techn.* in Lecture Notes in Computer Science. Berlin, Germany: Springer-Verlag, May 2003, pp. 614–629.

[18] E. Fujisaki and K. Suzuki, "Traceable ring signature," in *Public Key Cryptography—PKC* (Lecture Notes in Computer Science). Heidelberg, Germany: Springer, 2007, pp. 181–200.

[19] E. Fujisaki, "Sub-linear size traceable ring signatures without random oracles," in *Topics Cryptology—CT-RSA* (Lecture Notes in Computer Science). Heidelberg, Germany: Springer, 2011, pp. 393–415.

[20] B. Smyth, M. D. Ryan, and L. Chen, "Formal analysis of privacy in direct anonymous attestation schemes," *Sci. Comput. Program.*, vol. 111, pp. 300–317, Nov. 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167642315000702

[21] J. Guo, L. Hao, and H. Sun, "A new DAA scheme from one-off public key," in *Proc. Int. Conf. Electron., Commun. Control (ICECC)*, 2011, pp. 646–649.

[22] E. Brickell, L. Chen, and J. Li, "Simplified security notions of direct anonymous attestation and a concrete scheme from pairings," *Int. J. Inf. Secur.*, vol. 8, no. 5, pp. 315–330, Oct. 2009, doi: 10.1007/S10207-009-0076-3.

[23] L. Chen, D. Page, and N. P. Smart, "On the design and implementation of an efficient DAA scheme," in *Smart Card Research and Advanced Application* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2010, pp. 223–237.

[24] L. Chen, "A DAA scheme using batch proof and verification," in *Trust Trustworthy Computing* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2010, pp. 166–180.

[25] E. Brickell and J. Li, "Enhanced privacy ID: A direct anonymous attestation scheme with enhanced revocation capabilities," *IEEE Trans. Dependable Secure Comput.*, vol. 9, no. 3, pp. 345–360, May 2012.

[26] IoT Security. *A Cost-Effective Foundation for End-to-End.* Accessed: Feb. 24, 2022. [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/intel-epid-white-paper.pdf

[27] *Intel Enhanced Privacy ID (EPID) Security Technology*. Accessed: Feb. 24, 2022. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/intel-enhanced-privacy-id-epid-security-technology.html

[28] E. Brickell and J. Li, "Enhanced privacy ID from bilinear pairing," Tech. Rep. 095, 2009. [Online]. Available: http://eprint.iacr.org/2009/095

[29] J. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2015, pp. 281–310.

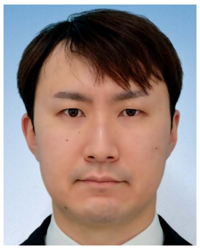
[30] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch, "Verifiable delay functions," in *Advances in Cryptology—CRYPTO* (Lecture Notes in Computer Science). New York, NY, USA: Springer, 2018, pp. 757–788.

[31] YCHARTS. *Bitcoin Average Transaction Fee.* Accessed: Apr. 5,2022. [Online]. Available: https://ycharts.com/indicators/bitcoin_average_transaction_fee

[32] T. Okamoto and K. Ohta, "Universal electronic cash," in *Advances in Cryptology—CRYPTO*, J. Feigenbaum, Ed. Heidelberg, Germany: Springer, 1992, pp. 324–337.

[33] D. Chaum and T. P. Pedersen, "Transferred cash grows in size," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science), vol. 658. Berlin, Germany: Springer, 1993, pp. 390–407, [Online]. Available: http://link.springer.com/10.1007/3-540-47555-9_32

[34] S. Canard, A. Gouget, and J. Traoré, "Improvement of efficiency in (unconditional) anonymous transferable E-cash," in *Financial Cryptography and Data Security* (Lecture Notes in Computer Science), vol. 5143. Heidelberg, Germany: Springer, 2008, pp. 202–214. [Online]. Available: http://link.springer.com/10.1007/978-3-540-85230-8_19
[35] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Advances in Cryptology—CRYPTO* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 1987, pp. 186–194.
[36] D. Pointcheval and J. Stern, "Security proofs for signature schemes," in *Advances in Cryptology—EUROCRYPT* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 1996, pp. 387–398.
[37] M. Fischlin, "Communication-efficient non-interactive proofs of knowledge with online extractors," in *Advances in Cryptology—CRYPTO* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2005, pp. 152–168.
[38] A. Dmitrienko, D. Noack, and M. Yung, "Secure wallet-assisted offline bitcoin payments with double-spender revocation," in *Proc. ACM Asia Conf. Comput. Commun. Secur.* New York, NY, USA: Association for Computing Machinery, 2017, pp. 1–12.
[39] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song, "Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, Jun. 2019, pp. 185–200.
[40] M. Bellare and C. Namprempre, "Authenticated encryption: Relations among notions and analysis of the generic composition paradigm," *J. Cryptol.*, vol. 21, no. 4, pp. 469–491, Oct. 2008, doi: 10.1007/S00145-008-9026-x.
[41] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2014, pp. 459–474.
[42] M. H. Au, W. Susilo, and Y. Mu, "Constant-size dynamic K-TAA," in *Security and Cryptography for Networks* (Lecture Notes in Computer Science). Berlin, Germany: Springer, 2006, pp. 111–125.



TAISEI TAKAHASHI received the master's degree in informatics from the Institute of Information Security, Japan, where he is currently pursuing the Ph.D. degree. His research interests include information security, especially blockchain and cryptography.

TAISHI HIGUCHI was born in 1989. He received the master's degree in physics from the Tokyo University of Science, Japan, Tokyo, in 2015. He is currently pursuing the Ph.D. degree with the Institute of Information Security (IISEC). He is also a Researcher at Sakura Information Systems Company Ltd. His research interests include the blockchain and cryptography.

AKIRA OTSUKA (Member, IEEE) was born in Osaka, in 1966. He received the B.E. and M.E. degrees from Osaka University, in 1989 and 1991, respectively, and the Ph.D. degree from The University of Tokyo, in 2002. Since 2002, he has been a Postdoctoral Fellow and a Cooperative Researcher with The University of Tokyo. From 2003 to 2005, he was a member of Cryptographic Technique Monitoring Subcommittee at CRYPTREC. Since 2005, he has been with the National Institute of Advanced Industrial Science and Technology (AIST). He worked as the Leader of Research Security Fundamentals, from 2006 to 2010. From 2007 to 2014, he was a Visiting Professor at Research and Development Initiative, Chuo University. Since 2017, he has been a Professor with the Graduate School of Information Security, Institute of Information Security.