Full version of an extended abstract published in PETS 2018.

Möbius: Trustless Tumbling for Transaction Privacy

Sarah Meiklejohn
University College London
`s.meiklejohn@ucl.ac.uk`

Rebekah Mercer
Aarhus University
`rebekah@cs.au.dk`

Abstract

Cryptocurrencies allow users to securely transfer money without relying on a trusted intermediary, and the transparency of their underlying ledgers also enables public verifiability. This openness, however, comes at a cost to privacy, as even though the pseudonyms users go by are not linked to their real-world identities, all movement of money among these pseudonyms is traceable. In this paper, we present Möbius, an Ethereum-based *tumbler* or *mixing service*. Möbius achieves strong notions of anonymity, as even malicious senders cannot identify which pseudonyms belong to the recipients to whom they sent money, and is able to resist denial-of-service attacks. It also achieves a much lower off-chain communication complexity than all existing tumblers, with senders and recipients needing to send only two initial messages in order to engage in an arbitrary number of transactions.

1 Introduction

When Bitcoin was initially deployed in 2009, it was heralded as an anonymous digital form of cash, as there are no credentials required to transfer funds (just as one does not need a bank account to use cash), and the *pseudonyms* that users go by within the system are not linked in any way to their real-world identities.

Despite these initial perceptions, it has now become clear that Bitcoin and other similarly structured cryptocurrencies are not anonymous, as the transparency of their transaction ledgers means that it is not only possible to track flows of bitcoins as they pass from one pseudonym to another, but also to cluster pseudonyms together and understand when funds are meaningfully changing hands [29, 31, 1, 24, 26, 28, 27, 33, 23]. In addition to this long line of research, there are also now companies (e.g., Chainalysis and Coinalytics) devoted solely to tracking bitcoins — and other cryptocurrencies that have the same structure — in order to perform risk scoring and aid law enforcement investigations.

Despite the arguable advantages of being able to track flows of funds, it is not desirable for anyone with access to the ledger to be able to do it at a wide scale, targeting both honest and dishonest users. For example, thieves could use clustering techniques to identify lucrative targets, and companies that pay their employees in bitcoin could use tracking techniques to spy on their spending habits. It is therefore just as important for honest users to be able to protect their privacy as it is for law enforcement to be able to track the activity of dishonest users.

Until a few years ago, the main option available to users who wished to improve upon the level of anonymity already present in Bitcoin was to use a *tumbler* (also known as a *mixing service*); i.e., a centralized service that would "clean" bitcoins: if you sent them your bitcoins then they would send you back bitcoins from another user, thus severing the link between the sender and recipient in the transaction. Due to their nature, however, such services are able to steal the bitcoins sent to them [24], and are furthermore reliant on having a high number of customers with roughly similar amounts they wish to transact in order to meaningfully prevent someone from linking the inputs and outputs [28, 12].

In response to this, the past few years have seen a large rise in the numbers of privacy-enhancing systems available for cryptocurrencies, whether it is Bitcoin-based centralized mixing services that cryptographically prevent the tumbler from stealing from its customers [8, 35, 17], Bitcoin-based "privacy overlays" that are effectively decentralized mixing protocols, in which users swap bitcoins themselves [22, 32], privacy-enhancing solutions for other cryptocurrencies [20], or entire standalone cryptocurrencies that are specifically designed to improve anonymity [3, 36].

Despite the broad availability of a wide variety of mixing services, most of them still suffer from some drawbacks; e.g., most decentralized mixing services require a high degree of coordination and allow every sender to learn the links between senders and recipients. In centralized mixing services, in the best case the tumbler is trusted for availability, as there is always a period of time where the user wishing to mix their coins has performed the transaction sending them to the mixer, but the tumbler has not yet sent them to the recipient. If the tumbler goes offline during this period, the sender will have lost control of their coins indefinitely. Finally, standalone cryptocurrencies must gain a significant following in order to have any value, and are not modular in the sense that if a flaw is found in the underlying protocol [26], its developers must update it using a cumbersome forking process.

Our contributions. To address the above limitations, we present Möbius, which achieves strong anonymity guarantees with minimal off-chain communication overhead and a minimal number of on-chain transactions. To accompany it, we also present (1) the first game-based security model for a tumbler, which is general enough to be of independent interest and allows us to cryptographically prove our security properties, and (2) an implementation that confirms that Möbius is efficient enough to be used in practice, with relatively cheap costs in terms of both computing transactions, and deploying and using the contract. In particular, the fact that participants in Möbius can perform an arbitrary number of transfers after the initial — and itself optional — exchange of just two messages means that they can use it in a purely non-interactive fashion (i.e., a sender can transfer value to a recipient without needing their permission or other type of engagement), just as they would use Bitcoin or any other standalone cryptocurrency.

Intuitively, Möbius replaces a centralized mixing service with an Ethereum smart contract that does the mixing autonomously; this allows it to resist availablity attacks, but without the high coordination costs or lack of modularity imposed by, respectively, decentralized solutions and standalone cryptocurrencies. After presenting Ethereum and the two cryptographic primitives we rely on, stealth addresses and ring signatures, in Section 2, we go on in Section 3 to outline our threat model and the cryptographic properties we want to achieve. We then present our construction of Möbius in Section 4 and evaluate both the abstract protocol design (in Section 5) and our prototype implementation (in Section 6). We then provide thorough comparisons with related work in Section 7. We finish by discussing several possible extensions and improvements to Möbius in Section 8, and concluding in Section 9.

2 Building Blocks

In this section, we describe the building blocks we need for Möbius: the Ethereum blockchain (Section 2.2), stealth addresses (Section 2.3), and linkable ring signatures (Section 2.4). In addition to providing the relevant background for each building block, we also provide formal definitions for both their input-output behavior and their security.

2.1 Preliminaries

If x is a binary string then $|x|$ denotes its bit length. If S is a finite set then $|S|$ denotes its size and $x \xleftarrow{\$} S$ denotes sampling a member uniformly from S and assigning it to x. $\lambda \in \mathbb{N}$ denotes the security parameter and 1^λ denotes its unary representation. ε denotes the empty string. For a tuple $t = (x_1, \ldots, x_n)$ we denote as $t[x_i]$ the value stored at x_i.

Algorithms are randomized unless explicitly noted otherwise. PT stands for polynomial-time. By $y \leftarrow A(x_1, \ldots, x_n; r)$ we denote running algorithm A on inputs $x_1, \ldots, x_n$ and random coins r and assigning its output to y. By $y \xleftarrow{\$} A(x_1, \ldots, x_n)$ we denote $y \leftarrow A(x_1, \ldots, x_n; r)$ for coins r sampled uniformly at random. By $[A(x_1, \ldots, x_n)]$ we denote the set of values that have positive probability of being output by A on inputs $x_1, \ldots, x_n$. Adversaries are modeled as algorithms.

We use games in definitions and proofs of security. A game G has a MAIN procedure whose output is the output of the game. $\Pr[\mathsf{G}]$ denotes the probability that this output is 1.

Central to all of our building blocks is the idea of a digital signature, which consists of three algorithms: via $(pk, sk) \xleftarrow{\$} \mathsf{DS.KeyGen}(1^\lambda)$ one generates a public and secret key; via $\sigma \xleftarrow{\$} \mathsf{DS.Sign}(sk, m)$ one uses the secret key to sign the message m; and via $0/1 \leftarrow \mathsf{DS.Verify}(pk, m, \sigma)$ one verifies whether or not σ represents a signature on the message m under the secret key associated with pk. For ease of exposition, we assume that either sk contains pk or that pk can be efficiently computed given sk. Due to its usage in most cryptocurrencies, we use ECDSA throughout, so the key generation algorithm for all our other primitives is the same as $\mathsf{DS.KeyGen}$; we thus use the unified term KeyGen to refer to all of them. KeyGen generally takes as input a group $\mathbb{G}$ of some prime-order q (where q is a λ-bit prime), but in what follows we treat this group as fixed given λ and refer to $\mathsf{KeyGen}(1^\lambda)$ rather than $\mathsf{KeyGen}(\mathbb{G})$.

2.2 Cryptocurrencies

The first cryptocurrency to be deployed was Bitcoin, which was introduced on January 3 2009. In Bitcoin, the ledger of transactions is stored in a *blockchain*, which is a chain of blocks where each block contains a list of transactions. In terms of the structure of Bitcoin transactions, users are identified by *addresses* (which are themselves hashes of public keys), and transactions essentially serve to transfer not bitcoins themselves, but the right to spend a certain amount of bitcoins from one party to another. This right is defined as belonging to list of *unspent transaction outputs* (or UTXOs for short), which specifies — as the name suggests — the transaction outputs that have not yet been used as input. A transaction can thus be thought of as a list of input addresses and output addresses, along with the value that should be sent to each output address. When a transaction is accepted into the ledger, the rights to spend the associated bitcoins transfer from the input addresses to the output addresses.

In contrast to this relatively simple transaction structure, which supports only atomic transfers of funds from one set of parties to another, the Ethereum blockchain acts as an (almost) Turing-complete distributed virtual machine (often referred to as the Ethereum Virtual Machine, or EVM), along with a built-in currency called ether. The increased functionality in Ethereum enables developers to create *smart contracts* on the blockchain, which are stateful programs that run autonomously. The only limitation on the functionality of smart contracts is their complexity: every operation they perform consumes a certain amount of *gas*, which is a subcurrency within Ethereum designed to limit the amount of computation that an individual contract can use.

Ethereum addresses take one of two forms: "externally owned" addresses, which behave like Bitcoin addresses and are similarly controlled by a secret key, or contract addresses, which store the immutable code of the smart contract. Contract addresses are also associated with some additional storage, which is used to store the state of the smart contract.

Ethereum transactions contain a destination address to, a signature σ authorizing the transaction (corresponding to a sender with keypair (pk, sk)), a gas limit (which, for ease of exposition, we ignore in our formal specifications below), an amount amt in ether, and an optional data field data. If the destination address is externally owned, then the transfer goes through just as it would in Bitcoin. If the destination address corresponds to a contract, then the contract code is executed (subject to the specified gas limit) on any inputs specified in the data field, which may result in updating the state of the contract and/or triggering additional transactions (subject to the gas limit, which in turn is restricted by the maximum gas limit per block).

Formally, with H a hash function, we define the cryptographic aspects of the formation and verification of generic blockchain transactions as follows: via $\mathsf{tx} \xleftarrow{\$} \mathsf{FormTx}(sk, \mathsf{to}, \mathsf{amt}, \mathsf{data})$ one creates a transaction, which involves creating $h \leftarrow H(\mathsf{to}, \mathsf{amt}, \mathsf{data})$ and $\sigma \xleftarrow{\$} \mathsf{DS.Sign}(sk, h)$ and returning $(pk, \sigma, h, \mathsf{to}, \mathsf{amt}, \mathsf{data})$; and via $0/1 \leftarrow \mathsf{VerifyTx}(\mathsf{tx})$ one verifies the signature in a transaction, which means returning $\mathsf{DS.Verify}(\mathsf{tx}[pk], \mathsf{tx}[h], \mathsf{tx}[\sigma])$.

2.3 Stealth addresses

In cryptocurrencies, *stealth addresses* (respectively, stealth keys) refer to addresses (respectively, keys) that have been derived from a master key, but that — without the value used to perform the derivation — cannot be linked to other addresses derived from the same key. The addresses generated by hierarchical deterministic (HD) wallets can be viewed as a form of stealth address [16], and as such these addresses are widely used in Bitcoin (and other cryptocurrencies) today.

In addition to the unified key generation algorithm KeyGen, we define the two algorithms associated with stealth addresses as follows: via $\mathsf{spk} \leftarrow \mathsf{SA.PubDerive}(\mathsf{mpk}, \mathsf{secret}, \mathsf{nonce})$ one can derive from a master public key and a shared secret and nonce a stealth public key; and via $\mathsf{ssk} \leftarrow \mathsf{SA.PrivDerive}(\mathsf{msk}, \mathsf{secret}, \mathsf{nonce})$ one can derive from a master secret key and a shared secret and nonce a stealth secret key.

Stealth keys then have the property that all keys derived from a valid master keypair are themselves a valid keypair; i.e., that for all $\mathsf{secret}, \mathsf{nonce} \in \{0,1\}^*$ and $(\mathsf{mpk}, \mathsf{msk}) \in [\mathsf{KeyGen}(1^\lambda)]$, $(\mathsf{SA.PubDerive}(\mathsf{mpk}, \mathsf{secret}, \mathsf{nonce}), \mathsf{SA.PrivDerive}(\mathsf{msk}, \mathsf{secret}, \mathsf{nonce})) \in [\mathsf{KeyGen}(1^\lambda)]$. In particular then, if two users Alice and Bob share a secret and a nonce, and Alice knows Bob's master public key, then by incrementing nonce Alice can create many transactions sending coins to many different stealth addresses, rather than to the same address every time.

For this to be meaningful, we need not only a notion of correctness, but also a notion of security; i.e., we want to ensure that stealth addresses cannot be linked by anyone who doesn't know the shared secret, even if they know both the master public key and the nonce. For that, we define a notion of *stealthiness* (which is really a form of re-randomizability) as follows:

Definition 2.1 (Stealthiness)**.** $(\mathsf{KeyGen}, \mathsf{SA.PubDerive}, \mathsf{SA.PrivDerive})$ *is* stealthy *if the distributions over derived keys and over randomly generated keys are equal; i.e., if for all* $\mathsf{secret}, \mathsf{nonce} \in \{0,1\}^*$ *and* $(\mathsf{mpk}, \mathsf{msk}) \in [\mathsf{KeyGen}(1^\lambda)]$,

$$\big\{(pk, \mathsf{nonce}, \mathsf{mpk}) \ \big|\ pk \leftarrow \mathsf{SA.PubDerive}(\mathsf{mpk}, \mathsf{secret}, \mathsf{nonce})\big\} = \big\{(pk, \mathsf{nonce}, \mathsf{mpk}) \ \big|\ (pk, sk) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)\big\}.$$

Bitcoin and Ethereum use ECDSA, which means KeyGen produces keys of the form (g^r, r), where g is the generator of a group $\mathbb{G}$ of prime order q and $r \in \mathbb{F}_q$. As used in these cryptocurrencies, the additional algorithms are then defined as follows:[1]

[1] In these algorithms we alter the input/output behavior slightly by having them output the incremented nonce as well. In what follows we use the version where they output just the key, for ease of exposition, but assume the nonce is incremented every time.

```
$\underline{\mathsf{SA.PubDerive}(\mathsf{mpk}, \mathsf{secret}, \mathsf{nonce})}$
$\mathsf{spk} \leftarrow \mathsf{mpk} \cdot g^{H(\mathsf{secret} \| \mathsf{nonce})}$
$\mathsf{nonce} \leftarrow \mathsf{nonce} + 1$
return $(\mathsf{spk}, \mathsf{nonce})$

$\underline{\mathsf{SA.PrivDerive}(\mathsf{msk}, \mathsf{secret}, \mathsf{nonce})}$
$\mathsf{ssk} \leftarrow \mathsf{msk} + H(\mathsf{secret} \| \mathsf{nonce})$
$\mathsf{nonce} \leftarrow \mathsf{nonce} + 1$
return $(\mathsf{ssk}, \mathsf{nonce})$
```

In this construction, it is clear that if secret is random and $H(\cdot)$ is modeled as a random oracle then spk is also distributed uniformly at random, so the derived keypair satisfies stealthiness. If we fix the length of the nonce then we can also prevent against length-extension attacks (although here these are prevented anyway due to the assumption about random oracles).

2.4 Ring Signatures

In contrast to regular digital signatures, which verify against a specific public key, ring signatures verify against a set, or ring, of public keys. This allows parties to prove that they are part of a group without revealing exactly which public key belongs to them [30]. In contrast to group signatures [10, 2, 9], in which users must register as members of the group and at any point in time its membership list is well defined, the ring used for ring signatures can be formed on an ad-hoc basis, which is especially useful in an open environment like a cryptocurrency. In addition to KeyGen, ring signatures involve the following two algorithms: via $\sigma \xleftarrow{\$} \mathsf{RS.Sign}(sk, R, m)$ one can use sk to sign m as a member of R; and via $0/1 \leftarrow \mathsf{RS.Verify}(R, m, \sigma)$ one can verify if a signature σ on a message m really came from a member of R.

In a regular ring signature, signatures reveal only that the signer is part of the ring, and nothing else. In a *linkable* ring signature scheme [21] (which is itself related to a *unique* ring signature [13, 14]), signatures instead reveal whether or not the signer has already produced a signature for that ring, although they still do not reveal their identity. This means that they require an extra algorithm as follows: via $0/1 \leftarrow \mathsf{RS.Link}(\sigma_1, \sigma_2)$ one can tell whether or not two signatures were produced by the same signer.

For completeness, we present formal security properties in Appendix A. Informally, the security properties of a linkable ring signature are as follows:

Anonymity: Modulo the ability to run $\mathsf{RS.Link}$, an adversary cannot identify which ring signature corresponds to which of the public keys in the ring;

Unforgeability: An adversary cannot produce a valid signature if it does not know a secret key corresponding to a public key included in the ring;

Exculpability: An adversary cannot produce a valid signature that links to the signature of another member of the ring, whose key the adversary does not control; and

Linkability: Any two signatures produced by the same signer within the same ring are publicly linkable (i.e., anyone can detect that they were produced by the same signer).

As we see in our construction in Section 4, linkable ring signatures allow us to construct a tumbler in which recipients remain anonymous but we can still ensure they only withdraw the funds to which they are entitled; i.e., they cannot withdraw funds twice by producing two ring signatures.

Concretely, we use a minimally adapted version of a linkable ring signature due to Franklin and Zhang [14] (FZ), whose security is implied by the Decisional Diffie-Hellman (DDH) assumption in the random oracle model. FZ signatures were chosen due to their compatibility with ECDSA, their efficient verification, and their reliance on a weak and well established assumption (in addition to the random oracle model, which we already assume anyway).

We modify slightly the role of the message within FZ signatures. As originally presented, FZ ring signatures can be linked only if they are produced by the same party over the same ring and message. To instead achieve the case where they can be linked if they are produced by the same party over the same ring (regardless of the message), we form the signature over the message exactly as given in FZ, but without including the message within the *linking tag*. In other words, the linking tag used in our modified FZ signature is formed as $H(R)^{x_i}$, rather than as $H(m\|R)^{x_i}$ (where m is the message, R is the ring, and x_i is the secret key of the i-th participant). This adapts the FZ signature from being a unique ring signature to one satisfying the standard definition of linkable ring signatures, without affecting any other security properties.

Sublinear linkable ring signatures (i.e., linkable ring signatures whose size is sublinear in the number of participants in the ring) would yield asymptotically smaller signatures, although it is not clear that they would necessarily be more efficient for the relatively small number of participants we expect in our setting. Moreover, all existing sublinear ring signatures require not only an accompanying linking tag to become linkable, but also — in order to maintain the same anonymity guarantees in the broader context of a tumbler — a zero-knowledge proof that the linking tag is correctly formed. Thus, while there do exist sublinear ring signatures based on the same security assumptions, such as the one by Groth and Kohlweiss [15], we do not consider them fully suitable for this setting. We discuss this further when we compare against existing solutions in Section 7.

3 Threat Model

This section describes the participants in the system and their interactions, both with one another and with the tumbler, and the goals we aim to achieve by using Möbius. In addition to providing an informal explanation of the setting, we also give formal algorithms and definitions of security, which may be independently useful in analyzing the security of other proposed tumblers.

3.1 Participants and interactions

Participants in Möbius can act as either *senders* or *recipients*, with senders taking on the responsibility of transferring funds to recipients via the tumbler. We often use Alice to refer to the sender, and Bob to refer to the recipient.

One of the main goals of Möbius is to minimize the off-chain communication required between the sender and recipient. We therefore consider only two interactions between them, one to initialize their interaction and one for Alice to (optionally) notify Bob that his funds are ready to be withdrawn. We also consider the interactions that each of them has with the tumbler, the checks that the tumbler performs, and the ways in which the tumbler updates its own internal state. In all the following interactions and algorithms, we assume the public state of the tumbler is implicitly given as input.

Initialize: The sender and the recipient, each in possession of their respective secret key sk_A and sk_B, (optionally) engage in this interaction to establish a shared basis for future transactions, which we denote $\mathsf{aux} \in \{0,1\}^*$.

$\mathsf{tx} \xleftarrow{\$} \mathsf{Deposit}(sk_A, pk_B, \mathsf{aux})$: The sender runs this algorithm to deposit a specific amount of funds into the tumbler.

$0/1 \leftarrow \mathsf{VerifyDeposit}(\mathsf{tx})$**:** The tumbler runs this algorithm to check that the sender's deposit is valid.

$\mathsf{ProcessDeposit}(\mathsf{tx})$**:** If the deposit is valid, the tumbler runs this algorithm to update its internal state accordingly.

Notify**:** The sender runs this algorithm to generate a notification that the funds are ready to be collected from the contract, which she can then send (off-chain) to the recipient.

$\mathsf{tx} \overset{\$}{\leftarrow} \mathsf{Withdraw}(sk_B, \mathsf{aux})$**:** The recipient runs this algorithm to withdraw his funds from the tumbler.

$0/1 \leftarrow \mathsf{VerifyWithdraw}(\mathsf{tx})$**:** The tumbler runs this algorithm to check that the recipient's withdrawal is valid.

$\mathsf{ProcessWithdraw}(\mathsf{tx})$**:** If the withdrawal is valid, the tumbler runs this algorithm to update its internal state accordingly.

3.2 Security goals

We consider security in terms of three goals: anonymity, availability, and theft prevention. Before defining security formally, we first define the oracles that we use in our games. Briefly, CORR allows an adversary to corrupt a recipient ℓ by learning his secret key. AD allows an adversary to deposit into a particular session j of a tumbler from a key under its control to any public key, and AW allows an adversary to arbitrarily withdraw from a session j. HD allows an adversary to instruct an honest sender i in a session j to deposit to a recipient ℓ, and HW allows it to instruct an honest recipient ℓ in a session j to withdraw. Formally, these oracles are defined as follows (with respect to C, the list of corrupted parties, H_d and H_w, the respective lists of honest deposits and withdrawals, and $\mathsf{tumblers}$, the list of contract identifiers associated with distinct sessions):

```
$\underline{\mathrm{AD}(\mathsf{tx}, j)}$
$b \leftarrow \mathsf{VerifyDeposit}(\mathsf{tumblers}[j], \mathsf{tx})$
if $(b)$ $\mathsf{ProcessDeposit}(\mathsf{tumblers}[j], \mathsf{tx})$
return $b$
```

```
$\underline{\mathrm{AW}(\mathsf{tx}, j)}$
$b \leftarrow \mathsf{VerifyWithdraw}(\mathsf{tumblers}[j], \mathsf{tx})$
if $(b)$ $\mathsf{ProcessWithdraw}(\mathsf{tumblers}[j], \mathsf{tx})$
return $b$
```

```
$\underline{\mathrm{CORR}(\ell)}$
add $pk_{B_\ell}$ to $C$
return $sk_{B_\ell}$
```

```
$\underline{\mathrm{HD}(i, j, \ell)}$
$\mathsf{tx} \overset{\$}{\leftarrow} \mathsf{Deposit}(sk_{A_i}, pk_{B_\ell})$
add $\mathsf{tx}$ to $H_d$
$\mathsf{ProcessDeposit}(\mathsf{tumblers}[j], \mathsf{tx})$
return $\mathsf{tx}$
```

```
$\underline{\mathrm{HW}(j, \ell)}$
if $(pk_{B_\ell} \notin \mathsf{tumblers}[j].\mathsf{keys}_B)$ return $\perp$
$\mathsf{tx} \overset{\$}{\leftarrow} \mathsf{Withdraw}(\mathsf{tumblers}[j], sk_{B_\ell})$
add $(j, \ell, \mathsf{tx})$ to $H_w$
$\mathsf{ProcessWithdraw}(\mathsf{tumblers}[j], \mathsf{tx})$
return $\mathsf{tx}$
```

In all our definitions that follow, we make reference to the number of participants in the contract, which we denote by n, and the recipient keys (or other identifiers) for which it has received deposits, which we denote by $\mathsf{tumbler.keys}_B$.

3.2.1 Anonymity

We would like to ensure that sender and recipient addresses are *anonymous*; i.e., that for a given sender, it is not possible to distinguish between their recipient and any other recipient using the tumbler. We consider this goal with respect to four types of attackers: (a) an eavesdropper who is acting as neither the sender nor the recipient; (b) a malicious sender (or set of senders); (c) a malicious recipient (or set of recipients); and (d) the tumbler itself. As we will see in Section 5, most existing solutions achieve anonymity with respect to (c) but not (b), whereas Möbius achieves anonymity with respect to (b) but not (c).

We define two variants of anonymity: *sender* anonymity and *recipient* anonymity. In the former, we allow an adversary to take over any actor in the system, but require that they still cannot distinguish between the deposits of two honest senders. We assume that deposits determine some identifier of the recipient, however, as a sender must specify in some way to whom they are sending money. We thus achieve this notion only in the case that the recipient is the same across the two senders. To define recipient anonymity with respect to malicious senders, we again allow an adversary to take over any actor in the system, but require that the withdrawal transactions produced by any two recipients (including ones that the adversary has placed deposits for, but not ones that it controls) should be indistinguishable. Formally, the definition for anonymity is as follows:

Definition 3.1. *Define* $\mathbf{Adv}^{d\text{-}anon}_{mix,\mathcal{A}}(\lambda) = 2\Pr[\mathsf{G}^{d\text{-}anon}_{mix,\mathcal{A}}(\lambda)] - 1$ *for* $d \in \{dep, with\}$*, where these games are defined as follows:*

```
MAIN $\mathsf{G}^{dep\text{-}anon}_{mix,\mathcal{A}}(\lambda)$
$(pk_i, sk_i) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)\ \forall i \in [n]$
$\mathsf{PK}_A \leftarrow \{pk_i\}_{i=1}^n;\ C, H_d, \mathsf{tumblers} \leftarrow \emptyset$
$b \xleftarrow{\$} \{0,1\}$
$(\mathsf{state}, j, pk, i_0, i_1) \xleftarrow{\$} \mathcal{A}^{\textsc{Corr},\text{AD},\text{HD},\text{AW}}(1^\lambda, \mathsf{PK}_A)$
$\mathsf{tx} \xleftarrow{\$} \mathsf{Deposit}(\mathsf{tumblers}[j], sk_{A_{\ell_b}}, pk)$
$b' \xleftarrow{\$} \mathcal{A}^{\textsc{Corr},\text{AD},\text{HD},\text{AW}}(\mathsf{state}, \mathsf{tx})$
$return\ (b' = b)$
```

```
MAIN $\mathsf{G}^{with\text{-}anon}_{mix,\mathcal{A}}(\lambda)$
$(pk_i, sk_i) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)\ \forall i \in [n]$
$\mathsf{PK}_B \leftarrow \{pk_i\}_{i=1}^n;\ C, H_w, \mathsf{tumblers} \leftarrow \emptyset$
$b \xleftarrow{\$} \{0,1\}$
$(\mathsf{state}, j, \ell_0, \ell_1) \xleftarrow{\$} \mathcal{A}^{\textsc{Corr},\text{AD},\text{AW},\text{HW}}(1^\lambda, \mathsf{PK}_B)$
$\mathsf{PK} \leftarrow \mathsf{tumblers}[j].\mathsf{keys}_B$
$if\ (pk_{B_{\ell_0}} \notin \mathsf{PK}) \vee (pk_{B_{\ell_1}} \notin \mathsf{PK})\ return\ 0$
$\mathsf{tx} \xleftarrow{\$} \mathsf{Withdraw}(\mathsf{tumblers}[j], sk_{B_{\ell_b}})$
$b' \xleftarrow{\$} \mathcal{A}^{\textsc{Corr},\text{AD},\text{AW},\text{HW}}(\mathsf{state}, \mathsf{tx})$
$if\ (pk_{\ell_b} \in C\ for\ b \in \{0,1\})\ return\ 0$
$if\ ((j, \ell_b, \cdot) \in H_w\ for\ b \in \{0,1\})\ return\ 0$
$return\ (b' = b)$
```

Then the tumbler satisfies sender anonymity *if for all PT adversaries* $\mathcal{A}$ *there exists a negligible function* $\nu(\cdot)$ *such that* $\mathbf{Adv}^{dep\text{-}anon}_{mix,\mathcal{A}}(\lambda) < \nu(\lambda)$*, and* recipient anonymity *if for all PT adversaries* $\mathcal{A}$ *there exists a negligible function* $\nu(\cdot)$ *such that* $\mathbf{Adv}^{with\text{-}anon}_{mix,\mathcal{A}}(\lambda) < \nu(\lambda)$.

3.2.2 Availability

We would like to prevent attacks on availability, meaning that (a) no one can prevent the sender from using the tumbler; and (b) once the money is in the tumbler, no one can prevent the honest recipient from withdrawing it. While the first property is not based on cryptographic aspects of the system, the second property is and thus we can model its security accordingly.

Formally, we consider an honest set of recipients that would like to withdraw even in the face of an adversary that can take on the role of any of the senders or additional recipients, and we say the adversary wins if they manage to get the tumbler into a state whereby an honest recipient is involved but unable to withdraw their funds. The formal definition for availability is as follows:

Definition 3.2. *Define* $\mathbf{Adv}^{avail}_{mix,\mathcal{A}}(\lambda) = \Pr[\mathsf{G}^{avail}_{mix,\mathcal{A}}(\lambda)]$*, where this game is defined as follows:*

```
MAIN $\mathsf{G}^{avail}_{mix,\mathcal{A}}(\lambda)$
$(pk_i, sk_i) \xleftarrow{\$} \mathsf{KeyGen}\left(1^\lambda\right) \ \ \forall i \in [n]$
$\mathsf{PK}_B \leftarrow \{pk_i\}_{i=1}^n;\ C, H_w, \leftarrow \emptyset$
$(\ell, j) \xleftarrow{\$} \mathcal{A}^{\text{Corr,AD,AW,HW}}\left(1^\lambda, \mathsf{PK}_B\right)$
$b \leftarrow \mathsf{VerifyWithdraw}(\mathsf{tumblers}[j], \mathsf{Withdraw}(sk_\ell))$
$\mathit{if}\ (pk_\ell \in C) \vee ((j, \ell, \cdot) \in H_w)\ \mathit{return}\ 0$
$\mathit{return}\ (b = 0) \wedge (pk_\ell \in \mathsf{tumblers}[j].\mathsf{keys}_B)$
```

Then the tumbler satisfies availability *if for all PT adversaries $\mathcal{A}$ there exists a negligible function $\nu(\cdot)$ such that $\mathbf{Adv}^{avail}_{mix,\mathcal{A}}(\lambda) < \nu(\lambda)$.*

3.2.3 Theft prevention

We would like to ensure that the scheme does not allow coins to be either withdrawn twice (as this will steal them from the last recipient who attempts to withdraw from the contract), or withdrawn by anyone other than the intended recipient. Formally, the definition for theft prevention is as follows:

Definition 3.3. *Define $\mathbf{Adv}^{theft}_{mix,\mathcal{A}}(\lambda) = \Pr[\mathsf{G}^{theft}_{mix,\mathcal{A}}(\lambda)]$, where this game is defined as follows:*

```
MAIN $\mathsf{G}^{theft}_{mix,\mathcal{A}}(\lambda)$
$(pk_i, sk_i) \xleftarrow{\$} \mathsf{KeyGen}\left(1^\lambda\right) \ \ \forall i \in [n]$
$\mathsf{PK}_B \leftarrow \{pk_i\}_{i=1}^n;\ C, H_w, \mathsf{contract} \leftarrow \emptyset$
$(\mathsf{tx}, j) \xleftarrow{\$} \mathcal{A}^{\text{Corr,AD,AW,HW}}\left(1^\lambda, \mathsf{PK}_B\right)$
$\mathit{if}\ (\mathsf{tumblers}[j].\mathsf{keys}_B \not\subseteq \mathsf{PK}_B \setminus C)\ \mathit{return}\ 0$
$\mathit{return}\ \mathsf{VerifyWithdraw}(\mathsf{tumblers}[j], \mathsf{tx})$
```

Then the tumbler satisfies theft prevention *if for all PT adversaries $\mathcal{A}$ there exists a negligible function $\nu(\cdot)$ such that $\mathbf{Adv}^{avail}_{mix,\mathcal{A}}(\lambda) < \nu(\lambda)$.*

4 Our Scheme: Möbius

4.1 Overview

Intuitively, Möbius replaces the central tumbler used in previous schemes with a mixing Ethereum smart contract, which allows it to run autonomously. This allows the system to achieve a strong notion of availability, and by combining stealth keys and ring signatures it also achieves anonymity, theft prevention, and very low communication overhead.

In shifting from a central tumbler to a smart contract stored on a public blockchain, there are several challenges we face. First, a central tumbler somewhat inherently relies on its ability to store some *secret information* that allows the tumbler, and only the tumbler, to release the funds once the rightful recipient attempts to claim them. In a public blockchain, this is impossible, as both the code of the smart contract and all its storage are globally visible. Second, many cryptographic operations (which are often needed in tumblers) require a source of *randomness.* The Ethereum Virtual Machine (EVM) is deterministic by its very nature: if it was not, state changes from contracts executed on different machines might not be consistent, which would cause nodes to fail to reach consensus. This means we cannot use any randomness in the smart contract.

To overcome these challenges, we leave all secret-key and randomized cryptographic operations to the sender and recipient to perform locally. First, to establish the ability to derive stealth keys, Alice and Bob share Bob's master public key, a secret, and a nonce. Every time she wishes to send a certain amount of money to Bob, Alice uses the shared secret to derive a fresh stealth public key from the master key.

If Alice were to directly send the money to the corresponding stealth address, then even if Alice and Bob never used the addresses involved again, the transaction would still create a link between them. Thus, to sever this link, Alice instead sends the money and the stealth key to a contract responsible for mixing this exact amount of money. (This means that if Alice wants to send an amount that is not covered by a single contract, she must first split her money into the correct denominations, as is done — for example — in traditional cryptographic e-cash. We discuss this further in Section 8.2.)

Once sufficiently many senders have paid into the contract, the list of stealth public keys it stores is used to form a ring. Bob can now reclaim his money in an anonymous fashion by forming a signature that verifies with respect to the ring in the contract. Here again, the naïve solution is problematic: if Bob forms a normal (unlinkable) ring signature, he could withdraw the funds of other recipients in the contract in addition to his own. If Bob withdraws to the same stealth address generated by Alice, then — because Alice's transaction reveals the link between her address and his stealth address — this would again trivially reveal the link between him and Alice. To address these issues, Bob instead creates a linkable ring signature to claim the funds (thus ensuring he can only withdraw once), and a new ephemeral address to receive the funds (thus ensuring that there is no link between the address that Alice uses to send the funds and the address at which he receives them).

It is important to acknowledge that the current version of the EVM does not allow newly created addresses without any value stored in them to send 0-ether transactions (which is what Bob does to withdraw from the contract). This will be changed in the next version of the EVM, Metropolis, but in the meantime we discuss ways to alter Möbius in order to maintain both trustlessness and compatibility with the current EVM in Section 8.5.

4.2 Initializing the contract

The contract, held at address id_{contract}, is initialized by specifying the amount in ether that it accepts and the threshold of senders that it wants to have. This means it is initialized with the following variables:

- $\mathsf{participants}$: number of parties needed to form the ring;
- amt: denomination of ether to be mixed;
- $\mathsf{pubkeys}[]$: the public keys over which the ring is formed;
- $\mathsf{senders}[]$: the sender addresses; and
- $\mathsf{sigs}[]$: the signatures seen thus far (used to check for double withdrawal attempts).

In addition to the code required to add public keys, addresses, and signatures to the appropriate lists, the contract also contains code to verify deposit and withdrawal transactions (see below). So as not to require on-chain storage of all signatures used in all previous transactions, the contract's storage is deleted after the last withdrawal takes place. (And as an optimization, we do not store the entire signature; see Section 6 for more information.)

4.3 Initializing Alice and Bob

In order for Alice to be able to send coins to Bob, she must first be aware of his master public key mpk_B. In the presence of an on-chain public-key directory, or if she otherwise has prior knowledge of mpk_B, they need only share the secret secret and initialize $\mathsf{nonce} \leftarrow 0$. If such a directory does not exist,

```
$\underline{\mathsf{Deposit}(sk_A, \mathsf{mpk}_B, \mathsf{secret}, \mathsf{nonce})}$
$\mathsf{spk}_B \leftarrow \mathsf{SA.PubDerive}\,(\mathsf{mpk}_B, \mathsf{secret}, \mathsf{nonce})$
return $\mathsf{FormTx}(sk_A, id_{\mathsf{contract}}, \mathsf{amt}, \mathsf{spk}_B)$

$\underline{\mathsf{VerifyDeposit}(\mathsf{tx})}$
if $(\mathsf{tx}[\mathsf{amt}] \neq id_{\mathsf{contract}}[\mathsf{amt}])$ return 0
if $(\mathsf{spk}_B \notin \mathbb{G}))$ return 0
return $\mathsf{VerifyTx}(\mathsf{tx})$

$\underline{\mathsf{ProcessDeposit}(\mathsf{tx})}$
add $\mathsf{addr}(\mathsf{tx}[pk])$ to $\mathsf{sendaddr}[]$
add $\mathsf{tx}[\mathsf{data}]$ to $\mathsf{pubkeys}[]$

$\underline{\mathsf{Withdraw}(\mathsf{msk}_B, \mathsf{secret}, \mathsf{nonce})}$
$R \leftarrow id_{\mathsf{contract}}[\mathsf{pubkeys}]$
$\mathsf{ssk}_B \leftarrow \mathsf{SA.PrivDerive}\,(\mathsf{msk}_B, \mathsf{secret}, \mathsf{nonce})$
$(pk_{\mathsf{ephem}}, sk_{\mathsf{ephem}}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$
$\sigma \xleftarrow{\$} \mathsf{RS.Sign}\,(\mathsf{ssk}_B, R, pk_{\mathsf{ephem}})$
return $\mathsf{FormTx}(pk_{\mathsf{ephem}}, sk_{\mathsf{ephem}}, id_{\mathsf{contract}}, 0, \sigma)$

$\underline{\mathsf{VerifyWithdraw}(\mathsf{tx})}$
$R \leftarrow id_{\mathsf{contract}}[\mathsf{pubkeys}]$
if $(\mathsf{RS.Verify}(R, \mathsf{tx}[pk], \mathsf{tx}[\mathsf{data}]) = 0)$ return 0
if $(\exists \sigma \in \mathsf{contract} : \mathsf{RS.Link}(\sigma, \mathsf{tx}[\mathsf{data}]) = 1)$ return 0
return $\mathsf{VerifyTx}(\mathsf{tx})$

$\underline{\mathsf{ProcessWithdraw}(\mathsf{tx})}$
add $\mathsf{tx}[\mathsf{data}]$ to $\mathsf{sigs}[]$
$\mathsf{tx}' \xleftarrow{\$} \mathsf{FormTx}(id_{\mathsf{contract}}, pk_{\mathsf{ephem}}, \mathsf{amt}, \varepsilon)$
```

Figure 1: The formal specification of the algorithms that make up Möbius.

the Initialize interaction must also serve to share their master public keys mpk_A and mpk_B, from which the secret secret can then be shared using elliptic curve Diffie-Hellman (ECDH) key exchange (or any other mechanism that results in a secret known only to Alice and Bob). This interaction thus enables both Alice and Bob to output $\mathsf{aux} = (\mathsf{secret}, \mathsf{nonce})$.

4.4 Paying in to the contract

To pay in to the contract, Alice first derives a new stealth public key for Bob. She then creates a transaction sent to the contract using amt as the value and the stealth public key as the data. Formally, Deposit is defined as in Figure 1 (where, as in what follows, we treat id_{contract} and amt as hard-coded).

(As illustrated here, and discussed further in Section 5, the use of stealth keys is largely a communication optimization, although it does also enable auditability, as we discuss in Section 8.3. If communication overhead were not a concern, the system would work the same if, instead of Alice deriving a new key for Bob in a non-interactive fashion, Bob simply sent a fresh key to Alice to use.)

Alice then broadcasts this transaction, which eventually reaches the contract. The contract now checks that this is a valid transaction; i.e., that it is formed correctly and contains the right amount for this contract, and also that the stealth public key is valid (i.e., has an associated secret key that can be used to withdraw from the contract). Formally, it runs VerifyDeposit, as defined in Figure 1. If these checks pass, it adds the relevant keys to the lists it maintains, as illustrated by ProcessDeposit in Figure 1.

When the required number of participants have joined, the smart contract broadcasts a notification, which is processed by Alice. Alice can then run Notify to tell Bob (off-chain) that the contract is ready and sends him the contract address id_{contract}. Alternatively, if the contracts are fixed and have been registered (or Bob otherwise knows id_{contract}), then Bob can process this notification himself and the extra message is not needed.

If some predefined time limit passes and not enough participants have joined the contract, there are two options we consider: either the contract uses $\mathsf{sendaddr[]}$ to refund the senders who have joined (which damages availability), or the contract goes through with the mix with the current number of participants (which reduces the size of the anonymity set). We leave it up to the creator of the contract which of these (or any other) solutions they want to implement.

4.5 Withdrawing from the contract

To withdraw from the contract, Bob fetches the ring description $\mathsf{pubkeys[]}$ from the contract. He then derives the stealth secret key associated with the stealth public key used by Alice, which allows him to create a ring signature to withdraw his funds from the contract and into an ephemeral address. Formally, $\mathsf{Withdraw}$ is defined as in Figure 1.

The contract checks that this is a valid transaction; i.e., that it contains a valid signature for the ring R defined by the contract, that it doesn't link to any signature previously used to withdraw from the contract, and that it is correctly formed. Formally, it runs $\mathsf{VerifyWithdraw}$, which is defined as in Figure 1.

If these checks pass, then the smart contract stores the signature in order to verify future withdrawals, and creates a transaction sending amt to $\mathsf{addr}(pk_{\mathsf{ephem}})$. Once all participants have withdrawn from the contract, it deletes all stored values except $\mathsf{participants}$ and amt to prepare itself for a new round of mixing.

Again, if some predefined time limit passes and one or more participants have not withdrawn from the contract, there are several options. First, one could simply not define such a time limit, and give recipients an arbitrary amount of time to withdraw their funds. Second, one could allow the creator or the contract to simply set the state back to the default, and claim any excess. Again, we leave this decision up to the creator of the contract.

As we elaborate on in Section 5, it is essential that $\mathsf{addr}(pk_{\mathsf{ephem}})$ is a freshly generated key. If the address corresponding to the public key spk_B is used, then an eavesropping adversary could simply hash all of the public keys deposited into the ring and see if any correspond to the addresses into which the funds are withdrawn. Because Alice's deposit transaction includes both her own address addr_A and spk_B, this will cause the sender and recipient addresses to be linked. It is important to observe, finally, that if a nonce is ever reused then stealthiness is lost.

5 Security

To demonstrate that Möbius satisfies the security model in Section 3, we prove the following theorem:

Theorem 5.1. *If the stealth address is secure (i.e., satisfies stealthiness) and the ring signature is secure (i.e., satisfies anonymity, linkability, unforgeability, and exculpability) then Möbius, as defined in Section 4, satisfies anonymity, availability, and theft prevention.*

To prove this theorem, we break it down into several lemmas. To formally prove the cryptographic aspects, we consider an adapted version of Möbius that does not use the optimization of using stealth addresses (i.e., instead has Bob send Alice a fresh key every time, as discussed in Section 4.4), and assumes that the set of possible recipient keys is fixed ahead of time. We consider this adapted version for ease of analysis, but the security of this adapted version in fact implies the security of the original version. Outside of these definitions, we discuss the security of the original version explicitly.

We discuss our assumptions about the Ethereum network where relevant below, but briefly we assume that attackers cannot perform denial-of-service attacks on both individual users (in the form of an eclipse attack [18]) or on the network as a whole.

We begin with anonymity. Here, we can show that both sender and recipient anonymity hold with respect to all malicious parties. In terms of joining the two notions together, however, overall anonymity does not hold with respect to malicious recipients, as the deposit reveals the link between addr_A and spk_B, and the recipient's own formation of his withdrawal transaction reveals the link between spk_B and pk_{ephem}. A malicious recipient can thus fully link their own sender and recipient. This can be somewhat mitigated if senders use ephemeral addresses to pay into the contract, although of course even this ephemeral address may be linked back to the address that was used to fund it.

Lemma 5.2. *If the stealth address is stealthy (Definition 2.1) and the ring signature is anonymous (Definition A.1), then Möbius satisfies recipient anonymity and (partial) sender anonymity with respect to all malicious parties (Definition 3.1), and overall anonymity with respect to malicious senders and eavesdroppers.*

Proof. A formal proof of recipient anonymity (Definition 3.1) can be found in Appendix B. Briefly, this proof relies on the anonymity of ring signatures to ensure that withdrawal transactions do not reveal any information about the recipient who formed them, and thus cannot be used to link spk_B to pk_{ephem}. This property holds with respect to malicious eavesdroppers, senders, and recipients (and trivially with respect to the tumbler). This places the recipient within the anonymity set of other participants in the contract, although we discuss in Section 8.4 ways to increase the size of the anonymity set via repeated use of contracts.

As for sender anonymity, it holds as long as the address addr_A used to pay into the contract is chosen uniformly at random, as deposits will be distributed identically across different senders. In practice, however, this address must contain some money, which in turn must have come from somewhere. We thus achieve this notion subject to the ability of an adversary to track the flow of money into this address, which depends on how it has been used.

As for overall anonymity (i.e., unlinkability), this is again something we cannot achieve with respect to malicious recipients. With respect to malicious eavesdroppers, we consider the original version of Möbius (i.e., the version that uses stealth addresses), and argue that it achieves the same level of anonymity as a non-optimized version of the system in which, rather than have Alice derive a stealth public key, Bob simply generates a fresh keypair and (out of band) sends the corresponding public key to Alice. By stealthiness, the distributions of these two versions are identical. As the second game furthermore reveals no information about the identity of the recipient, we achieve this stronger notion of anonymity with respect to eavesdroppers.

If at least two senders are honest, then in fact the same argument applies with respect to the other senders: they will not be able to tell which sender is paying which recipient. If instead an adversary fills in all but one slots in a contract, then this aspect of sender anonymity is trivially broken. We cannot definitively prevent this from happening, but (as previous solutions suggest) if both deposits and withdrawals require a fee then we can disincentivize this behavior by making it prohibitively expensive.

Finally, we mention that because the contract waits until it receives a certain number of participants, and because all public keys in the ring are one-time use, we reduce our vulnerability to statistical analysis such as intersection or timing attacks [11] as well. □

Next, we show availability, which is broken down into two parts: first, the requirement that no one can prevent a sender from depositing into the contract, and second, the requirement that no one can prevent an honest recipient from withdrawing their own money from the contract. We cannot prove the first property cryptographically (due to its relationship to non-cryptographic aspects of the system), but nevertheless argue that it holds under reasonable assumptions.

Lemma 5.3. *If the ring signature satisfies exculpability (Definition A.4), Möbius satisfies availability (Definition 3.2).*

Proof. Briefly, we consider attacks on the first property as carried out by four classes of adversaries: (a) the tumbler, (b) other potential senders, (c) other potential recipients, and (d) the creator of the mixing contract.

For (a), whereas attacks on availablity could trivially be carried out in previous schemes using central services, by the tumbler simply going offline, in Möbius the mixing contract executes autonomously so this cannot happen.

For (b), if all but one parties on the network refuse to use the contract indefinitely, then — according to the choices we present in Section 4.4 — either the one remaining sender will have their deposit refunded, or their transaction will go through in a non-anonymous fashion. In the latter case there is no damage to availability (although it negates the point of using the contract), and in the former case the sender has wasted the gas costs associated with forming the transaction, but at least has not wasted additional resources (like a fee, or significant time spent in off-chain coordination), or revealed any information about either herself or her intended recipient.

For (c), other potential recipients have no control over whether or not a deposit is placed, so cannot prevent a sender from doing so.

For (d), once the creator has deployed the contract, they cannot prevent transfers from going through, as the contract executes autonomously. The only exception is if the creator optionally adds the ability to invoke the Ethereum `suicide` function, which would allow them to kill the smart contract, but this is visible in the contract so if it is included then users can at least make themselves aware of this risk. If they wish to go further, users can choose to deploy their own version of the contract without such a function.

A formal proof of the cryptographic aspect of the second property (i.e., that Definition 3.2 holds) is in Appendix C. Briefly, we argue that an adversary can prevent an honest recipient's withdrawal transaction from verifying only if it propagates the contract with another transaction that includes a ring signature linking to the one of the honest recipient. This violates exculpability.

Non-cryptographically, if all but one parties refuse to withdraw from the contract, this does not affect any recipient's ability to withdraw, nor does it affect the size of his anonymity set (which is fixed as the number of participants in the contract). The only way to prevent an honest recipient from withdrawing funds stored in the contract would be to stop all transactions from being accepted into the ledger (i.e., perform a denial-of-service attack on the whole Ethereum network), or perform a persistent man-in-the-middle attack on Bob in order to intercept his withdrawal transactions before they reach the network. We consider these attacks expensive enough that they are unlikely to happen. □

Finally, we show theft prevention.

Lemma 5.4. *If the ring signature satisfies linkability (Definition A.3) and unforgeability (Definition A.2), then Möbius satisfies theft prevention (Definition 3.3).*

A proof of this lemma can be found in Appendix D. Briefly, if an adversary has not corrupted any recipients but can nevertheless provide a withdrawal transaction that verifies, there are three possibilities: it came up with the transaction entirely by itself, it derived the transaction from a withdrawal transaction it witnessed an honest recipient perform, or it intercepted the transaction before it was included in a block (a "front-running" attack) and replaced the recipient address with one of its own. We can argue that the first and third cases violate unforgeability, and the second case violates linkability.

Given our specific choices of stealth address and ring signature (presented in Section 2) and putting everything together, we obtain the following corollary:

Corollary 5.5. *If $H(\cdot)$ is a random oracle and DDH holds in $\mathbb{G}$ then Möbius, as defined in Section 4, satisfies anonymity, availability, and theft prevention.*

6 Implementation

To ensure that Möbius is efficient enough to be used in practice, we implemented the system, and in this section provide various performance benchmarks for it, both in terms of the time taken for off-chain computations and the gas consumed in on-chain computations. We use the current gas price of 1 Gwei/gas,[2][3] and the average ether price over November 2017 of 346 USD/ether to calculate the dollar price of participating in the mix. Our performance benchmarks were collected on a laptop with an Intel Core i7 2.5GHz CPU and 8 GB of RAM.

6.1 Implementation details

The mixing contract consists of roughly 350 lines of Solidity code, a high-level language targeting the Ethereum Virtual Machine (EVM). Solidity is compiled to EVM byte code, which is then broadcast to the network and replicated in every node's storage. The functions VerifyDeposit, ProcessDeposit, VerifyWithdraw, and ProcessWithdraw consume gas (as they are performed on-chain), but the others are off-chain and thus free. The code for both the sender and the recipient consists of roughly 400 lines of Go code, and allows them to (respectively) generate ECDSA stealth addresses and Franklin-Zhang (FZ) ring signatures [14].

Using the native support for elliptic curve cryptography and big number arithmetic on the EVM, we implemented in Solidity the functions required for the contract to run VerifyDeposit and VerifyWithdraw. Briefly, ecmul computes the product of an elliptic curve point and a 256-bit scalar over bn256; ecadd computes the sum of two elliptic curve points; expmod computes exponentials of the form $b^e \bmod m$, where m is an integer of up to 256 bits; and get_y computes the y-coordinate associated with a given x-coordinate. The gas costs for these functions are shown in Table 1.

In our current implementation, the sender provides the full stealth public key, which the contract then stores in pubkeys[]. This is because the current gas costs in Ethereum make it cheaper to store the entire key, as opposed to having the sender send only the x-coordinate of spk_B and then have the contract recompute the corresponding y-coordinate when needed.

The version of Möbius presented in Section 4 required the contract to store ring signatures in sigs[], and verify against these when verifying additional signatures. Using FZ signatures, we can significantly reduce this cost: as briefly discussed in Section 2.4, FZ signatures consist of both a zero-knowledge proof of membership and a *linking tag*, which for the i-th party is formed as $H(R)^{x_i}$ (where R is the ring and x_i is the secret key). To perform the linking necessary to check for double withdrawals, the contract therefore need only store the tags, rather than the full signatures.

Finally, for ease of development our current implementation stores all contract values in Ethereum. To further reduce storage costs, however, we could store only a reference hash in the contract storage and then store the data itself in IPFS,[4] which is much cheaper in terms of storage.

6.2 Costs

We now consider the costs, in terms of computation, storage, and gas, of each of the operations in our system. These are summarized in Tables 1 and 2. In terms of latency, this is dependent on the number of participants in the contract (which also determines the size of the anonymity set, and the cost of a withdrawal transaction), so we leave this as a parameter n and highlight particular choices of n where appropriate.

[2] 1 GWei is 1×10^{-9} ether, with Wei being the smallest denomination of ether.

[3] `https://ethgasstation.info/`

[4] `https://ipfs.io/`

Function		Cost (gas)	Cost (USD)
ecmul		42,018	0.015
ecadd		2668	0.0009
get_y		14,415	0.0049
expmod		3428	0.0011
deposit	$i = 1$	105,346	0.036
	$i > 1$	76,123	0.026
withdraw		335,714n	0.116n

Table 1: The costs, in both gas and USD, for our elliptic curve functions, and for the two functions needed to process transactions. For deposit, i denotes the number of the sender paying in (i.e., if they are the first sender or not). n denotes the number of participants in the mix. A constant 38,403 gas (0.013 USD) is added to all withdraw transactions. The values in USD are computed using the November 2017 costs of 1 GWei/gas and 346 USD/ether.

An Ethereum transaction consists of the following fields: a nonce, which for our purposes is always 1 byte; gasPrice and amt in Wei, represented by their hex encoding; gasLimit, which is also hex encoded; the to address, which is 20 bytes; data, with 32 bytes per argument; and the ECDSA signature over the hash of all the above, which is 65 bytes. (One can recover the sender's public key from the signature, which is why it is not given explicitly.) In function calls, the function signature, which is formed by taking the first 4 bytes of the hash of the function name and parameters, is prepended to data. This means that the base size of an Ethereum transaction sending 1 ether to a function is 120 bytes.

The stealth public key that the sender must include in their deposit transaction is 64 bytes, so a deposit transaction is 184 bytes. On the recipient side, we do not need to include the ring description (as it is the pubkeys[] field of the contract), but must include the intended recipient address (or its corresponding public key), which also functions as the message being signed. The ring signature is only $64(n + 1)$ bytes, where n is the size of the ring, and the message is an additional 64 bytes, as the data field pads all values to 32 bytes. This means the withdraw transaction is $120 + 64(n + 2) = 248 + 64n$ bytes (and, for example, we need to get to $n = 12$ before the transaction size exceeds 1 kB.)

The contract then needs to store pubkeys[], which means storing $64n$ bytes. It also needs to store sendaddr[], which means storing $20n$ bytes. Finally, storing sigs[] means storing (at most) $64n$ bytes, given the optimization of storing only tags we discussed above. In total then, the contract must store up to $148n$ bytes. At the current cost of 20,000 gas per 32 bytes of storage, this requires $100{,}000n$ gas, the cost of which is distributed between the senders and recipients. (Again, to provide concrete numbers, this means we need to get to $n = 28$ before the storage cost exceeds 1 USD.)

In Ethereum, the cost to deploy a contract is a fixed 32,000 gas, plus an amount determined by the length and complexity of the contract [37]. The initial cost of deploying the Möbius mixing contract is 1,046,027 gas (0.36 USD). The contract needs to be deployed only once, by anyone who wants to do so, and can then be reused arbitrarily many times.

Table 1 shows the gas costs of the main functions in the elliptic curve arithmetic library, and the two functions used in the ring mixing contract that forms Möbius. For deposit, it currently uses more gas (105,346) to process the transaction of the first sender paying in, as it costs extra gas to set storage in the EVM, but after the first sender the costs are the same (76,123). For withdraw, the costs reflect the ring signature verification and thus are dependent on the number of participants n. In terms of comparing these costs, the current suggested fee for an average Bitcoin transactions is 38,420 satoshis.[5] As of November 2017, with Bitcoin worth 11,379 USD (or as of the current writing much more), this is

[5]`https://bitcoinfees.earn.com/`

Function	Time taken (ms)
deposit	0.098
withdraw	3.254

Table 2: Time taken, in milliseconds, to generate the transactions needed in Möbius, for $n = 4$. The time for deposit is averaged over 3,000 runs, and the time for withdraw over 5,000 runs.

	Anonymity against...			Availability		Theft prevention
	outsiders	senders	recipients	sender	tumbler	
Centralized						
Mixcoin [8]	TTP*	✗	✓	✓	✗	TTP
Blindcoin [35]	✓	✗	✓	✓	✗	TTP
TumbleBit [17]	✓	✓	✓	✓	✗	✓
Decentralized						
Coinjoin [22]	✓	✗	✓	✗	n.a.	✓
Coinshuffle [32]	✓	✗	✓	✗	n.a.	✓
XIM [7]	✓	✗	✓	✓	n.a.	✓
Möbius	✓	✓	✗	✓	✓	✓

Table 3: The security properties achieved by each system discussed in this section. For anonymity, we consider security against the three adversaries mentioned in Section 3: an outside eavesdropper, a sender, and a recipient. For availability, we consider the ability of other participants to cause a sender to waste their effort in a transaction that does not go through, and also consider the ability of a tumbler itself to prevent mixing (which is not applicable in decentralized systems). TTP denotes a trusted third party.
* In Mixcoin the third party learns the mapping, but under the assumption they do not reveal it, the system achieves anonymity against an outside observer.

4.37 USD, so we can see all our operations are substantially cheaper. While all values in USD should be taken with a grain of salt, due to the volatility of both Bitcoin and Ethereum, this means that even if we use $n = 350$ then Möbius is roughly the same price as one of the most efficient existing tumblers, TumbleBit [17], and while the latency may be higher, the overall coordination and computational costs are still much lower.

In terms of off-chain computations for the parties, Table 2 gives the computation time for the sender to run Deposit and for the recipient to run Withdraw. Again, we see that the costs are very minimal.

7 Related Work

We consider as related work any solution that is attempting to improve anonymity in cryptocurrencies beyond what is provided by the basic usage of pseudonyms, either in terms of mixing services, so-called "privacy overlays," or standalone cryptocurrencies. Möbius achieves different anonymity but stronger overall security guarantees relative to related work, and is overall more efficient.

	# Off-chain messages	# Transactions
Centralized		
Mixcoin [8]	2	2
Blindcoin [35]	4	2
TumbleBit [17]	12	4
Decentralized		
Coinjoin [22]	$O(n^2)$	1
Coinshuffle [32]	$O(n)$	1
XIM [7]	0	7*
Möbius	2**	2

Table 4: The communication complexity, in terms of off-chain messages and on-chain transactions, required for each system. For the centralized ones we capture the cost for a single sender-recipient pair (because they can mix through the tumbler), while for the decentralized ones we capture the cost for n participants to mix (as they can mix only with each other).
* XIM works between two participants, and so cannot be fully compared with the other decentralized mixes, in which n parties can participate.
** In our system, Möbius, the exchange (consisting of two messages) is needed only once per pair, not per transaction, so the cost can be amortized across many transactions.

7.1 Tumblers and mixing solutions

Table 3 presents a comparison of the security properties of previously proposed tumblers [8, 35, 22, 32, 7, 17], both centralized and decentralized, and Table 4 presents an efficiency comparison.

Looking at Table 3, we can see that Möbius essentially achieves the opposite anonymity guarantees to all other schemes. In particular, while all schemes achieve anonymity against passive eavesdroppers, all of them except TumbleBit require the recipient to send the sender their address in the clear — as is done in a standard Bitcoin payment — in order to initiate the transfer, rendering anonymity with respect to malicious senders impossible.

Beyond anonymity, we also see that Möbius achieves different availability guarantees from previous systems. In centralized systems, the tumbler could go down either before the transfer takes place, or — worse — the tumbler could go down during the transfer (after Alice has sent coins to the tumbler but before it has sent them to Bob), in which case the coins are lost. In these systems, tumblers are therefore very much relied upon for availability. On the other hand, they can achieve low latency, as participants do not need to rely on others to mix.

Current decentralized systems, in contrast, have higher latency due to the coordination costs needed to set up the mix. They also enable malicious senders to waste the resources of other senders, by pretending to engage in the transaction and going offline only at the last minute. XIM [7] minimizes this risk by requiring participants to pay a fee upfront, but — as we see in Table 4 — this comes at the cost of a high number of transactions and thus high latency. Möbius inherits some of the latency issues of decentralized solutions, but requires no coordination amongst senders and achieves a high level of availability. It also mitigates the risk of senders wasting each other's time because senders commit to participating in the mix and release their funds in the same atomic transaction.

In Table 4, we can see that Möbius uses significantly fewer combined messages and transactions than any previous system, as even for Mixcoin there are still 2 off-chain messages required per transfer (rather than our amortized cost). We also see a clear trade-off between centralized systems, which typically require more rounds of interaction throughout but achieve constant communication complexity (as all parties can communicate directly with the tumbler), and decentralized systems, in which parties need to coordinate amongst themselves. The exception is XIM, which has the explicit goal of avoiding all off-chain communication, but comes at the cost of requiring a high number of on-chain transactions [7].

7.2 Standalone cryptocurrencies

While we cannot compare directly against standalone cryptocurrencies, as they operate in a different model (e.g., there is no notion of theft prevention) and are not designed as modular solutions, we nevertheless compare anonymity and efficiency properties.

In Monero [36], senders select several other unspent transaction outputs, all of which must hold the same value, and forms a ring signature over them to achieve anonymity for the sender. This type of passive participation also provides users with plausible deniability. Monero does not, however, achieve anonymity for recipients, as just like for decentralized mixes senders need to know their address in order to create the transaction; again, this seems inherent in any solution with a single transaction. Furthermore, in terms of security, coin forgery holds only computationally in Monero, an attack on its anonymity was possible until recently [26], and it is still subject to timing-related attacks. In terms of efficiency, due to the growing size of the set of unspent transaction outputs, Monero addresses are 69 bytes (in contrast to Ethereum's 20-byte addresses), and the average transaction size is 13 kB, which is larger than a Möbius transaction with 160 participants.

Zcash[6] is based on the notion of succinct zero-knowledge proofs (zk-SNARKs for short), which were first proposed for use on the blockchain in 2013 [25, 3] and allow users to prove that a given transaction is valid without revealing any specific information about it (e.g., the sender and recipient, or the value being sent) [19]. In terms of security, Zcash achieves essentially the strongest notion of anonymity possible, as the anonymity set is all other Zcash users who engage in *shielded* transactions (so-called transparent transactions, in which no anonymity is achieved, are also possible in Zcash). Despite recent advances [4, 5], however, Zcash requires a trusted setup, without which a malicious party could forge an arbitrary number of coins. In terms of efficiency, transaction generation is currently extremely expensive, taking an average of 48 seconds to generate and using 4 GB of RAM.[7]

Finally, while not a full standalone cryptocurrency, Möbius is also related to Zerocoin [3], and in particular to the simplified version of Zerocoin presented by Groth and Kohlweiss [15], which is made possible by their sublinear ring signature. In this setting, (non-linkable) ring signatures are accompanied by a serial number, known to both the sender and recipient, which achieves a weaker notion of anonymity than Möbius (as the sender and recipient can link themselves to the other). While this could be modified (as discussed in Section 2.4) to achieve the same level of anonymity by providing instead a zero-knowledge proof of knowledge of the serial number, this would be at the cost of efficiency.

8 Extensions and Improvements

8.1 Minimizing communication

The off-chain communication costs in Möbius are already fairly minimal, with the sender and recipient needing to agree on just the shared secret in Initialize. As the channel over which they communicate this secret does not need to be secure, in theory the communication could be moved on-chain, with the sender and recipient broadcasting their respective mpk_A and mpk_B and constructing secret using

[6]`https://z.cash`

[7]`https://speed.z.cash`

ECDH. There are also options for eliminating the interaction in Initialize entirely, such as adding an on-chain mpk directory.

The notification step could also be eliminated, as previously discussed, by having a directory, or registry, of mixing contracts. Identifying whether or not sufficient parties have joined the contract is available in our implementation through a function checkn, which returns the number of participants currently in the ring for a given mixing contract, and the desired threshold is available as part of the contract state. This function consumes no gas, as it is simply a query, but requires the intended recipient to know id_{contract}.

8.2 Sending arbitrary denominations

In the existence of a contract registry, or even without one, one could also allow a sender Alice to be able to send any denomination of funds to a recipient Bob (albeit at a relatively high cost). This would be achieved by, for example, deploying and registering a contract for every power of two, and then having Alice perform a base-two decomposition of her desired denomination and send the appropriate amount to the appropriate contract, using a different stealth address for Bob each time.

Depending on the value of ether, this process could of course be prohibitively expensive in terms of both transaction fees and gas costs. It therefore would also be useful to have additional contracts for common denominations, or to investigate other decompositions.

8.3 Auditability

In addition to reducing the off-chain communication overhead, the deterministic nature of stealth keys also makes it possible for a third party to audit the system, as long as senders and recipients share their auxiliary information with such a party (thus allowing them to derive stealth keys and observe their use within previous contracts). This means that Möbius may be useful not only on the public Ethereum platform, but also in permissioned blockchain settings. For example, if transfers are permitted only to known recipients with known (or 'whitelisted') master public keys, then when recipients withdraw they can also include a ring signature over the list of authorized master public keys, thus proving that they are a known recipient. This allows participants to operate in a controlled environment but still retain financial privacy.

8.4 Increasing anonymity

In Möbius, the size of the anonymity set depends on the number of participants in a given contract. Given the tradeoff between the size of this set and the efficiency of the mix (both in terms of the latency required for each participant, as they wait for other senders to join, and in terms of the cost of using linear ring signatures), senders may not get the level of anonymity they want in a single iteration of Möbius.

If the level of anonymity gained through use of one tumbler is not sufficient, users can chain tumblers to build a mix network similar to those in Mixcoin [8] and Blindcoin [35]. Thus, rather than having the recipient spend the funds they receive in a normal transaction, they are instead fed directly into another session of Möbius. This further obfuscates the flow of money, and the size of the anonymity set grows as a sum of the anonymity sets in each session.

8.5 EVM compatibility

Currently, Ethereum allows calls only from addresses that already hold ether, so can pay for gas with the account balance. Giving contracts the ability to pay for gas used in the execution of contract calls is a planned change in Ethereum, but Möbius as presented is currently incompatible with Ethereum, as our specification of Withdraw requires contract calls to be sent from freshly generated addresses (with zero balance).

One way to work around this issue in the meantime is to outsource the FormTx aspect of Withdraw to accounts that do hold ether. In this case, the intended recipients would still generate an ephemeral keypair and form the ring signature including as a message the address they intend the ether to be withdrawn to. The recipient would then send the ring signature and address to the party responsible for running FormTx, rather than executing this themselves. Due to the ring signature being formed over the ephemeral address, we ensure that the party to whom the withdrawal transaction is outsourced has no ability to withdraw the funds to their own account, and thus does not need to be trusted. To incentivize this party or service, the recipient can also pay them a fee when the funds are released.

9 Conclusions

We have proposed Möbius, a low-cost mixing system based on smart contracts that provides participants with not only strong notions of privacy, but also strong availability due to the resilience of the Ethereum blockchain. Due to the functionality of smart contracts, no coordination is required between parties wishing to transact in the same mix, no trust is required in off-chain servers, and no parties (even when all but one collude) have the ability to stop any honest party's funds from being withdrawn by the intended recipient. Even between parties wishing to transfer funds to each other, the communication cost is minimal and we have discussed — among other possible extensions to the system — ways to eliminate this overhead entirely. Given its practicality, and the modifications we describe to make it compatible with the EVM, we believe that Möbius could be deployed today.

Acknowledgments

Sarah Meiklejohn is supported in part by EPSRC Grant EP/N028104/1, and Rebekah Mercer was supported in part by Clearmatics Technologies, Ltd. and in part by the Danish Independent Research Council under Grant-ID DFF-6108-00169 (FoCC).

References

[1] E. Androulaki, G. Karame, M. Roeschlin, T. Scherer, and S. Capkun. Evaluating user privacy in Bitcoin. In A.-R. Sadeghi, editor, *FC 2013*, volume 7859 of *LNCS*, pages 34–51, Okinawa, Japan, Apr. 1–5, 2013. Springer, Heidelberg, Germany.

[2] M. Bellare, D. Micciancio, and B. Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In E. Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 614–629, Warsaw, Poland, May 4–8, 2003. Springer, Heidelberg, Germany.

[3] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from Bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474, Berkeley, CA, USA, May 18–21, 2014. IEEE Computer Society Press.

[4] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza. SNARKs for C: Verifying program executions succinctly and in zero knowledge. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 90–108, Santa Barbara, CA, USA, Aug. 18–22, 2013. Springer, Heidelberg, Germany.

[5] E. Ben-Sasson, A. Chiesa, M. Green, E. Tromer, and M. Virza. Secure sampling of public parameters for succinct zero knowledge proofs. In *2015 IEEE Symposium on Security and Privacy*, pages 287–304, San Jose, CA, USA, May 17–21, 2015. IEEE Computer Society Press.

[6] A. Bender, J. Katz, and R. Morselli. Ring signatures: Stronger definitions, and constructions without random oracles. In S. Halevi and T. Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 60–79, New York, NY, USA, Mar. 4–7, 2006. Springer, Heidelberg, Germany.

[7] G. Bissias, A. P. Ozisik, B. N. Levine, and M. Liberatore. Sybil-resistant mixing for Bitcoin. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, pages 149–158. ACM, 2014.

[8] J. Bonneau, A. Narayanan, A. Miller, J. Clark, J. A. Kroll, and E. W. Felten. Mixcoin: Anonymity for Bitcoin with accountable mixes. In N. Christin and R. Safavi-Naini, editors, *FC 2014*, volume 8437 of *LNCS*, pages 486–504, Christ Church, Barbados, Mar. 3–7, 2014. Springer, Heidelberg, Germany.

[9] X. Boyen and B. Waters. Full-domain subgroup hiding and constant-size group signatures. In T. Okamoto and X. Wang, editors, *PKC 2007*, volume 4450 of *LNCS*, pages 1–15, Beijing, China, Apr. 16–20, 2007. Springer, Heidelberg, Germany.

[10] D. Chaum and E. van Heyst. Group signatures. In D. W. Davies, editor, *EUROCRYPT'91*, volume 547 of *LNCS*, pages 257–265, Brighton, UK, Apr. 8–11, 1991. Springer, Heidelberg, Germany.

[11] G. Danezis and A. Serjantov. Statistical disclosure or intersection attacks on anonymity systems. In *International Workshop on Information Hiding*, pages 293–308. Springer, 2004.

[12] J. Edwards. Two guys on Reddit are chasing a thief who has $220 million in bitcoins, Dec. 2013. `www.businessinsider.com/220-million-sheep-marketplace-bitcoin-theft-chase-2013-12`.

[13] M. Franklin and H. Zhang. A framework for unique ring signatures. Cryptology ePrint Archive, Report 2012/577, 2012. `http://eprint.iacr.org/2012/577`.

[14] M. K. Franklin and H. Zhang. Unique ring signatures: A practical construction. In A.-R. Sadeghi, editor, *FC 2013*, volume 7859 of *LNCS*, pages 162–170, Okinawa, Japan, Apr. 1–5, 2013. Springer, Heidelberg, Germany.

[15] J. Groth and M. Kohlweiss. One-out-of-many proofs: Or how to leak a secret and spend a coin. In E. Oswald and M. Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 253–280, Sofia, Bulgaria, Apr. 26–30, 2015. Springer, Heidelberg, Germany.

[16] G. Gutoski and D. Stebila. Hierarchical deterministic bitcoin wallets that tolerate key leakage. In R. Böhme and T. Okamoto, editors, *FC 2015*, volume 8975 of *LNCS*, pages 497–504, San Juan, Puerto Rico, Jan. 26–30, 2015. Springer, Heidelberg, Germany.

[17] E. Heilman, L. Alshenibr, F. Baldimtsi, A. Scafuro, and S. Goldberg. TumbleBit: an untrusted Bitcoin-compatible anonymous payment hub. In *Proceedings of NDSS 2017*, 2017.

[18] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg. Eclipse attacks on Bitcoin's peer-to-peer network. In *Proceedings of USENIX Security 2015*, 2015.

[19] D. Hopwood, S. Bowe, T. Hornby, and N. Wilcox. Zcash Protocol Specification, Version 2016.0-alpha-3.1, May 2016. Accessed at: `https://github.com/zcash/zips/blob/master/protocol/protocol.pdf`.

[20] A. E. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE Symposium on Security and Privacy*, pages 839–858, San Jose, CA, USA, May 22–26, 2016. IEEE Computer Society Press.

[21] J. K. Liu, V. K. Wei, and D. S. Wong. Linkable spontaneous anonymous group signature for ad hoc groups (extended abstract). In H. Wang, J. Pieprzyk, and V. Varadharajan, editors, *ACISP 04*, volume 3108 of *LNCS*, pages 325–335, Sydney, NSW, Australia, July 13–15, 2004. Springer, Heidelberg, Germany.

[22] G. Maxwell. CoinJoin: Bitcoin privacy for the real world. `bitcointalk.org/index.php?topic=279249`, Aug. 2013.

[23] S. Meiklejohn and C. Orlandi. Privacy-enhancing overlays in Bitcoin. In M. Brenner, N. Christin, B. Johnson, and K. Rohloff, editors, *FC 2015 Workshops*, volume 8976 of *LNCS*, pages 127–141, San Juan, Puerto Rico, Jan. 30, 2015. Springer, Heidelberg, Germany.

[24] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G. M. Voelker, and S. Savage. A fistful of bitcoins: characterizing payments among men with no names. In *Proceedings of the 2013 Internet Measurement Conference*, pages 127–140. ACM, 2013.

[25] I. Miers, C. Garman, M. Green, and A. D. Rubin. Zerocoin: Anonymous distributed E-cash from Bitcoin. In *2013 IEEE Symposium on Security and Privacy*, pages 397–411, Berkeley, CA, USA, May 19–22, 2013. IEEE Computer Society Press.

[26] A. Miller, M. Möser, K. Lee, and A. Narayanan. An empirical analysis of linkability in the Monero blockchain. *arXiv preprint arXiv:1704.04299*, 2017.

[27] P. Moreno-Sanchez, M. B. Zafar, and A. Kate. Listening to whispers of Ripple: Linking wallets and deanonymizing transactions in the Ripple network. *Proceedings on Privacy Enhancing Technologies*, 2016(4):436–453, 2016.

[28] M. Möser, R. Böhme, and D. Breuker. An inquiry into money laundering tools in the Bitcoin ecosystem. In *Proceedings of the APWG E-Crime Researchers Summit*, 2013.

[29] F. Reid and M. Harrigan. An analysis of anonymity in the Bitcoin system. In *Security and privacy in social networks*, pages 197–223. Springer, 2013.

[30] R. L. Rivest, A. Shamir, and Y. Tauman. How to leak a secret. In C. Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 552–565, Gold Coast, Australia, Dec. 9–13, 2001. Springer, Heidelberg, Germany.

[31] D. Ron and A. Shamir. Quantitative analysis of the full Bitcoin transaction graph. In A.-R. Sadeghi, editor, *FC 2013*, volume 7859 of *LNCS*, pages 6–24, Okinawa, Japan, Apr. 1–5, 2013. Springer, Heidelberg, Germany.

[32] T. Ruffing, P. Moreno-Sanchez, and A. Kate. CoinShuffle: Practical decentralized coin mixing for Bitcoin. In M. Kutylowski and J. Vaidya, editors, *ESORICS 2014, Part II*, volume 8713 of *LNCS*, pages 345–364, Wroclaw, Poland, Sept. 7–11, 2014. Springer, Heidelberg, Germany.

[33] M. Spagnuolo, F. Maggi, and S. Zanero. BitIodine: Extracting intelligence from the Bitcoin network. In N. Christin and R. Safavi-Naini, editors, *FC 2014*, volume 8437 of *LNCS*, pages 457–468, Christ Church, Barbados, Mar. 3–7, 2014. Springer, Heidelberg, Germany.

[34] P. P. Tsang and V. K. Wei. Short linkable ring signatures for e-voting, e-cash and attestation. Cryptology ePrint Archive, Report 2004/281, 2004. `http://eprint.iacr.org/2004/281`.

[35] L. Valenta and B. Rowan. Blindcoin: Blinded, accountable mixes for Bitcoin. In M. Brenner, N. Christin, B. Johnson, and K. Rohloff, editors, *FC 2015 Workshops*, volume 8976 of *LNCS*, pages 112–126, San Juan, Puerto Rico, Jan. 30, 2015. Springer, Heidelberg, Germany.

[36] N. van Saberhagen. Cryptonote v 2.0. Published at `https://cryptonote.org/whitepaper.pdf`, 2013.

[37] G. Wood. Ethereum: A Secure Decentralised Generalised Transaction Ledger – Homestead Revision, Jan. 2016. `gavwood.com/paper.pdf`.

A Formal Definitions for Linkable Ring Signatures

In this section, we give formal versions of the four security properties — anonymity, unforgeability, exculpability, and linkability — for linkable ring signatures from Section 2.4. For all our definitions, we use the following two oracles (defined with respect to two sets C and S):

```
$\underline{\textsc{Corr}(i)}$
add $pk_i$ to $C$
return $sk_i$

$\underline{\textsc{Sign}(i, R, m)}$
if $pk_i \notin R$ return $\bot$
$\sigma \xleftarrow{\$} \mathsf{RS.Sign}(sk_i, R, m)$
add $(i, R, m)$ to $S$
return $\sigma$
```

Informally, anonymity says that an adversary should not be able to tell which one of two signers created a given signature. Given that signatures from the same signer can be linked, we cannot satisfy the strongest version of this definition [6, Definition 4], but instead use a modified version from Franklin and Zhang for the case of linkable ring signatures [13].

Definition A.1 (Anonymity). *Define* $\mathbf{Adv}^{anon}_{rs,\mathcal{A}}(\lambda) = 2\Pr[\mathsf{G}^{rs,\mathcal{A}}_{anon}(\lambda)] - 1$*, where this game is defined as follows:*

```
$\underline{\text{MAIN } \mathsf{G}^{rs,\mathcal{A}}_{anon}(\lambda)}$
$(pk_i, sk_i) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)\ \forall i \in [n];\ \mathsf{PK} \leftarrow \{pk_i\}_{i=1}^n$
$b \xleftarrow{\$} \{0,1\};\ C, S \leftarrow \emptyset$
$(\mathsf{state}, i_0, i_1, R, m) \xleftarrow{\$} \mathcal{A}^{\text{CORR,SIGN}}(\mathsf{PK})$
$\sigma^* \xleftarrow{\$} \mathsf{RS.Sign}\,(sk_{i_b}, R, m)$
$b' \xleftarrow{\$} \mathcal{A}^{\text{CORR,SIGN}}(\mathsf{state}, \sigma^*)$
$return\ (b = b') \wedge (sk_{i_\beta} \notin C) \wedge (i_\beta, R, \cdot) \notin S)\ for\ \beta \in \{0,1\}$
```

Then the ring signature is anonymous *if for all PT adversaries $\mathcal{A}$ there exists a negligible function $\nu(\cdot)$ such that $\mathbf{Adv}^{forge}_{rs,\mathcal{A}}(\lambda) < \nu(\lambda)$.*

Informally, unforgeability says that only users who possess one of the secret keys corresponding to one of the public keys in the ring are able to create valid ring signatures. Here we use the strongest version of this definition [6].

Definition A.2 (Unforgeability)**.** *Define $\mathbf{Adv}^{forge}_{rs,\mathcal{A}}(\lambda) = \Pr[\mathsf{G}^{rs,\mathcal{A}}_{forge}(\lambda)]$, where this game is defined as follows:*

```
$\underline{\text{MAIN } \mathsf{G}^{rs,\mathcal{A}}_{forge}(\lambda)}$
$(pk_i, sk_i) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)\ \forall i \in [n];\ \mathsf{PK} \leftarrow \{pk_i\}_{i=1}^n$
$C, S \leftarrow \emptyset$
$(R, m, \sigma) \xleftarrow{\$} \mathcal{A}^{\text{CORR,SIGN}}(\mathsf{PK})$
$return\ \mathsf{RS.Verify}(R, m, \sigma) \wedge ((\cdot, R, m) \notin S) \wedge (R \subseteq \mathsf{PK} \setminus C)$
```

Then the ring signature is unforgeable *if for all PT adversaries $\mathcal{A}$ there exists a negligible function $\nu(\cdot)$ such that $\mathbf{Adv}^{forge}_{rs,\mathcal{A}}(\lambda) < \nu(\lambda)$.*

Informally, linkability says that any two signatures produced by the same party must be identifiable are such. Here we use an adapted version of the definition by Tsang and Wei [34, Definition 5]. In the game, $\mathcal{A}$ wins if it knows only one secret key associated with the ring (either via corruption or by choosing the key itself), but can nevertheless produce two valid signatures that don't link. In short, we require here that either $\mathcal{A}$ can query the signing oracle once, it can corrupt one honestly generated public key, or it can generate one key adversarially.

Definition A.3 (Linkability)**.** *Define $\mathbf{Adv}^{link}_{rs,\mathcal{A}}(\lambda) = \Pr[\mathsf{G}^{rs,\mathcal{A}}_{link}(\lambda)]$, where this game is defined as follows:*

```
$\underline{\text{MAIN } \mathsf{G}^{rs,\mathcal{A}}_{link}(\lambda)}$
$(pk_i, sk_i) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)\ \forall i \in [n];\ \mathsf{PK} \leftarrow \{pk_i\}_{i=1}^n$
$C, S \leftarrow \emptyset$
$(m_1, m_2, , R, \sigma_1, \sigma_2) \xleftarrow{\$} \mathcal{A}^{\text{CORR,SIGN}}(\mathsf{PK})$
$if\ (\mathsf{RS.Verify}(R, m_b, \sigma_b) = 0)\ for\ b = 1 \vee b = 2\}\ return\ 0$
$c \leftarrow |\{s \in S\ :\ s = (\cdot, R, \cdot)\}|$
$if\ (c > 1)\ return\ 0$
$if\ (c = 0)\ return\ (|R \cap C| + |R \setminus \mathsf{PK}| \le 1) \wedge (\mathsf{RS.Link}(\sigma_1, \sigma_2) = 0)$
$return\ (R \subseteq \mathsf{PK} \setminus C) \wedge (\mathsf{RS.Link}(\sigma_1, \sigma_2) = 0)$
```

Then the ring signature is linkable *if for all PT adversaries $\mathcal{A}$ there exists a negligible function $\nu(\cdot)$ such that $\mathbf{Adv}^{link}_{rs,\mathcal{A}}(\lambda) < \nu(\lambda)$.*

Finally, exculpability says that no colluding set of parties can produce a signature that links to the signature of an honest party. In the game we define, $\mathcal{A}$ picks a user within a specified ring who it would like to frame, and gets a signature from that user. It wins if, via corrupting other users and folding its own users into the ring, it can produce another signature that links to that of the honest user.

Definition A.4 (Exculpability)**.** *Define* $\mathbf{Adv}^{frame}_{rs,\mathcal{A}}(\lambda) = \Pr[\mathsf{G}^{rs,\mathcal{A}}_{frame}(\lambda)]$*, where this game is defined as follows:*

```
$\underline{\text{MAIN } \mathsf{G}^{rs,\mathcal{A}}_{frame}(\lambda)}$
$(pk_i, sk_i) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)\ \forall i \in [n];\ \mathsf{PK} \leftarrow \{pk_i\}_{i=1}^n$
$C, S \leftarrow \emptyset$
$(\mathsf{state}, R, i, m) \xleftarrow{\$} \mathcal{A}^{\text{CORR},\text{SIGN}}(\mathsf{PK})$
$\sigma^* \xleftarrow{\$} \mathsf{RS.Sign}(sk_i, R, m)$
$\sigma \xleftarrow{\$} \mathcal{A}^{\text{CORR},\text{SIGN}}(\mathsf{state}, \sigma^*)$
$if\ (pk_i \in C) \vee (pk_i \notin R) \vee ((i, R, \cdot) \in S)\ return\ 0$
$return\ \mathsf{RS.Verify}(R, m, \sigma) \wedge \mathsf{RS.Link}(\sigma^*, \sigma)$
```

Then the ring signature is exculpable *if for all PT adversaries $\mathcal{A}$ there exists a negligible function $\nu(\cdot)$ such that* $\mathbf{Adv}^{frame}_{rs,\mathcal{A}}(\lambda) < \nu(\lambda)$.

B Proof of Anonymity

Let $\mathcal{A}$ be a PT adversary playing $\mathsf{G}^{\text{with-anon}}_{\text{mix},\mathcal{A}}(\lambda)$. We build an adversary $\mathcal{B}$ such that

$$\mathbf{Adv}^{\text{with-anon}}_{\text{mix},\mathcal{A}}(\lambda) \leq \mathbf{Adv}^{\text{anon}}_{\text{rs},\mathcal{B}}(\lambda),$$

from which Lemma 5.2 follows.

Briefly, $\mathcal{B}$ uses the set of public keys it is given as the set of recipient public keys to give to $\mathcal{A}$. On CORR and HW queries, it uses its own oracles (CORR and SIGN, respectively) to answer them, and on AD and AW queries $\mathcal{B}$ behaves honestly. When $\mathcal{A}$ outputs its challenge (j, ℓ_0, ℓ_1), $\mathcal{B}$ outputs its own challenge using ℓ_0, ℓ_1, $\mathsf{tumblers}[j].\mathsf{keys}_B$ as the ring, and an ephemeral key pk_{ephem} as the message. It then embeds the resulting signature into a withdrawal transaction, gives that to $\mathcal{A}$, and outputs whatever $\mathcal{A}$ does. The code for $\mathcal{B}$ (omitting the oracles in which it behaves honestly) is as follows:

```
$\underline{\mathcal{B}^{\text{CORR,SIGN}}(1^\lambda, \mathsf{PK})}$
$H_w \leftarrow \emptyset$
$(\mathsf{state}, j, \ell_0, \ell_1) \xleftarrow{\$} \mathcal{A}^{\text{CORR,AD,AW,HW}}(1^\lambda, \mathsf{PK})$
$(pk_{\mathsf{ephem}}, sk_{\mathsf{ephem}}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$
$\sigma \xleftarrow{\$} \text{CHAL}(\ell_0, \ell_1, \mathsf{tumblers}[j].\mathsf{keys}_B, pk_{\mathsf{ephem}})$
$(pk_{\mathsf{ephem}}, sk_{\mathsf{ephem}}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$
$\mathsf{tx} \xleftarrow{\$} \mathsf{FormTx}(sk_{\mathsf{ephem}}, \mathsf{tumblers}[j], 0, \sigma)$
$b' \xleftarrow{\$} \mathcal{A}^{\text{CORR,AD,AW,HW}}(\mathsf{state}, \mathsf{tx})$
return $b'$

$\underline{\text{CORR}(\ell)}$
return $\text{CORR}(\ell)$

$\underline{\text{HW}(\ell, j)}$
$R \leftarrow \mathsf{tumblers}[j].\mathsf{keys}_B$
if $(pk_{B_\ell} \notin R)$ return $\perp$
$(pk_{\mathsf{ephem}}, sk_{\mathsf{ephem}}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$
$\sigma \xleftarrow{\$} \text{SIGN}(R, \ell, pk_{\mathsf{ephem}})$
$\mathsf{tx} \leftarrow \mathsf{FormTx}(sk_{\mathsf{ephem}}, \mathsf{tumblers}[j], 0, \sigma)$
add $(j, \ell, \mathsf{tx})$ to $H_w$
$\mathsf{ProcessWithdraw}(\mathsf{tumblers}[j], \mathsf{tx})$
return $\mathsf{tx}$
```

By the first winning condition of the tumbler anonymity game, we know that if $\mathcal{A}$ wins then it hasn't corrupted ℓ_0 or ℓ_1, then (because $\mathcal{B}$ corrupts users only when $\mathcal{A}$ does) the second winning condition of the ring signature anonymity game is met. By the second winning condition, we know that if $\mathcal{A}$ hasn't queried the HW oracle on ℓ_b and j for either $b \in \{0, 1\}$ then (again, because $\mathcal{B}$ queries its SIGN oracle only when $\mathcal{A}$ queries HW) similarly $(\ell_b, R, \cdot) \notin S$, so the third winning condition of the ring signature anonymity game is met. Finally, $\mathcal{B}$ outputs the same guess bit as $\mathcal{A}$, so if $\mathcal{A}$ guesses correctly (i.e., the third winning condition of Definition 3.1 is met) then so will $\mathcal{B}$, which means the first winning condition of its game is met as well.

C Proof of Availability

Let $\mathcal{A}$ be a PT adversary playing $\mathsf{G}^{\text{avail}}_{\text{mix},\mathcal{A}}(\lambda)$. We build an adversary $\mathcal{B}$ such that

$$\mathbf{Adv}^{\text{avail}}_{\text{mix},\mathcal{A}}(\lambda) \leq \mathbf{Adv}^{\text{frame}}_{\text{rs},\mathcal{B}}(\lambda),$$

from which Lemma 5.3 follows.

Briefly, $\mathcal{B}$ first uses its given set of public keys are the recipient public keys to give to $\mathcal{A}$. When $\mathcal{A}$ queries the AD oracle, $\mathcal{B}$ behaves honestly, and when it queries the AW oracle it also behaves honestly but keeps track of the ring signatures used by $\mathcal{A}$. When $\mathcal{A}$ queries the CORR oracle, $\mathcal{B}$ queries its own CORR oracle and returns the result, and when $\mathcal{A}$ queries the HW oracle $\mathcal{B}$ answers using its SIGN oracle. At the end of the game, $\mathcal{B}$ challenges the oracle on the user and tumbler keys specified by $\mathcal{A}$ to receive back the honest ring signature. It then outputs a signature that had been previously used by $\mathcal{A}$, which we can argue must be linked to the challenge signature. Formally, the code for $\mathcal{B}$ is as follows:

```
$\underline{\mathcal{B}^{\text{Corr},\text{Sign}}(1^\lambda, \mathsf{PK})}$
$\Sigma, H_w \leftarrow \emptyset$
$(\ell, j) \xleftarrow{\$} \mathcal{A}^{\text{Corr},\text{AD},\text{AW},\text{HW}}(1^\lambda, \mathsf{PK})$
$(pk_{\mathsf{ephem}}, sk_{\mathsf{ephem}}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$
$\sigma^* \xleftarrow{\$} \text{Chal}(\mathsf{tumblers}[j].\mathsf{keys}_B, \ell, pk_{\mathsf{ephem}})$
find $\sigma \in \Sigma[j]$ such that $\mathsf{RS.Link}(\sigma^*, \sigma) = 1$
return $\sigma$

$\underline{\text{Corr}(\ell)}$
return $\text{Corr}(\ell)$

$\underline{\text{AW}(\mathsf{tx}, j)}$
$b \leftarrow \mathsf{VerifyWithdraw}(\mathsf{tumblers}[j], tx)$
if $(b)$
    add $\mathsf{tx}[\mathsf{data}]$ to $\Sigma[j]$
    $\mathsf{ProcessWithdraw}(\mathsf{tumblers}[j], \mathsf{tx})$
return $b$

$\underline{\text{HW}(j, \ell)}$
$R \leftarrow \mathsf{tumblers}[j].\mathsf{keys}_B$
if $(pk_{B_\ell} \notin R)$ return $\perp$
$(pk_{\mathsf{ephem}}, sk_{\mathsf{ephem}}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$
$\sigma \xleftarrow{\$} \text{Sign}(R, \ell, pk_{\mathsf{ephem}})$
$\mathsf{tx} \leftarrow \mathsf{FormTx}(sk_{\mathsf{ephem}}, \mathsf{tumblers}[j], 0, \sigma)$
add $(j, \ell, \mathsf{tx})$ to $H_w$
$\mathsf{ProcessWithdraw}(\mathsf{tumblers}[j], \mathsf{tx})$
return $\mathsf{tx}$
```

It is clear, given that $\mathcal{A}$ makes only polynomially many queries to each oracle, that Σ is at most polynomial size so can be traversed efficiently by $\mathcal{B}$ to find σ. To argue why such a σ must exist, we consider the first winning condition for $\mathcal{A}$ in Definition 3.2, which says that the honest withdrawal transaction for pk_{B_ℓ} does not verify. Looking at the code for VerifyWithdraw in Figure 1, we can rule out the possibility that RS.Verify or VerifyTx do not pass (as the transaction was honestly formed, so by correctness they will), which leaves us with one option: there is a signature σ stored in $\mathsf{tumblers}[j]$ such that $\mathsf{RS.Link}(\sigma, \mathsf{tx}[\mathsf{data}]) = 1$. As $\mathcal{B}$ obtains the honest signature from pk_{B_ℓ} as its challenge, it therefore must be able to find such a σ in order for $\mathcal{A}$'s winning condition to hold. Furthermore, because σ is such that $\mathsf{RS.Link}(\sigma^*, \sigma) = 1$ (by definition of how it is chosen from Σ) and $\mathsf{RS.Verify}(R, pk_{\mathsf{ephem}}, \sigma) = 1$ (by definition of how it is added to Σ), the fourth and fifth winning conditions of the exculpability game (Definition A.4) are also satisfied.

To argue for the first three winning conditions, we observe that they align exactly with the winning conditions of the availability game: $pk_\ell \notin C$ holds by $\mathcal{A}$'s second winning condition (because the corrupted sets are identical), $pk_\ell \in R$ holds because of the fifth winning condition of the availability game, and $(\ell, R, \cdot) \notin S$ holds by $\mathcal{A}$'s third winning condition, as if $\mathcal{B}$ queried its Sign oracle on $(R, \ell, pk_{\mathsf{ephem}})$ for $R = \mathsf{tumblers}[j].\mathsf{keys}_B$ then $(j, \ell, \cdot) \in H_w$.

D Proof of Theft Prevention

Let $\mathcal{A}$ be a PT adversary playing $\mathsf{G}^{\text{theft}}_{\text{mix},\mathcal{A}}(\lambda)$. We build adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$ such that

$$\mathbf{Adv}^{\text{theft}}_{\text{mix},\mathcal{A}}(\lambda) \leq \mathbf{Adv}^{\text{forge}}_{\text{rs},\mathcal{B}_1}(\lambda) + \mathbf{Adv}^{\text{link}}_{\text{rs},\mathcal{B}_2}(\lambda), \tag{1}$$

from which Lemma 5.4 follows.

We consider two cases: either all the winning conditions of Definition 3.3 hold and there is no entry of the form $(j, \cdot, \cdot) \in H_w$ (an event we denote $E_{\notin}$), or all the winning conditions hold and there is at least one entry of the form $(j, \cdot, \cdot) \in H_w$ (an event we denote $E_{\in}$). It is clear that

$$\mathbf{Adv}^{\text{theft}}_{\text{mix},\mathcal{A}}(\lambda) = \Pr[E_{\notin}] + \Pr[E_{\in}].$$

We deal with $E_{\notin}$ first, and construct $\mathcal{B}$ such that

$$\Pr[E_{\notin}] \leq \mathbf{Adv}^{\text{forge}}_{\text{rs},\mathcal{B}}(\lambda). \tag{2}$$

Briefly, $\mathcal{B}_1$ first uses its given set of public keys as the set of recipient public keys to give to $\mathcal{A}$. When $\mathcal{A}$ queries the AD and AW oracles, $\mathcal{B}_1$ behaves honestly (i.e., runs the oracles as specified in Section 3). When $\mathcal{A}$ queries the Corr oracle, $\mathcal{B}_1$ queries its own Corr oracle and returns the result. When $\mathcal{A}$ queries the HW oracle, $\mathcal{B}_1$ first extracts the ring of public keys from the state of the tumbler, and uses this to query its own Sign oracle to get the ring signature. It then follows the honest code to embed this signature into a withdrawal transaction and returns the results to $\mathcal{A}$. At the end of the game, $\mathcal{B}_1$ extracts the signature from the transaction output by $\mathcal{A}$ and outputs that, along with the associated ring. Formally, the code for $\mathcal{B}_1$ (omitting the oracles in which it behaves honestly) is as follows:

```
$\underline{\mathcal{B}_1^{\text{Corr,Sign}}(1^\lambda, \mathsf{PK})}$
$H_w \leftarrow \emptyset$
$(\mathsf{tx}, j) \xleftarrow{\$} \mathcal{A}^{\text{Corr,AD,AW,HW}}(1^\lambda, \mathsf{PK})$
return $(\mathsf{tumblers}[j].\mathsf{keys}_B, \mathsf{tx}[pk], \mathsf{tx}[\mathsf{data}])$

$\underline{\text{Corr}(\ell)}$
return $\text{Corr}(\ell)$

$\underline{\text{HW}(j, \ell)}$
$R \leftarrow \mathsf{tumblers}[j].\mathsf{keys}_B$
if $(pk_\ell \notin R)$ return $\perp$
$(pk_{\mathsf{ephem}}, sk_{\mathsf{ephem}}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$
$\sigma \xleftarrow{\$} \text{Sign}(R, \ell, pk_{\mathsf{ephem}})$
$\mathsf{tx} \leftarrow \mathsf{FormTx}(sk_{\mathsf{ephem}}, \mathsf{tumblers}[j], 0, \sigma)$
add $(j, \ell, \mathsf{tx})$ to $H_w$
$\mathsf{ProcessWithdraw}(\mathsf{tumblers}[j], \mathsf{tx})$
return $\mathsf{tx}$
```

By the first winning condition of the theft prevention game, we know that $\mathsf{tumblers}[j].\mathsf{keys}_B \subseteq \mathsf{PK}_B \setminus C$. Given that $\mathcal{B}_1$ uses $R \leftarrow \mathsf{tumblers}[j].\mathsf{keys}_B$ and that the sets C are the same across the two games, this implies that the third winning condition of Definition A.2 is met. By the second winning condition of the theft prevention game, we know that $\mathsf{VerifyWithdraw}(\mathsf{tx}, \mathsf{tumblers}[j]) = 1$. Looking at the code for VerifyWithdraw in Figure 1, this further implies that $\mathsf{RS.Verify}(R, \mathsf{tx}[pk], \mathsf{tx}[\mathsf{data}]) = 1$, which means the first winning condition of the unforgeability game is met. Finally, by the assumption that $(j, \cdot, \cdot) \notin H_w$, we know that $\mathcal{B}_1$ did not query its SIGN oracle on $(R, \cdot, \cdot)$, which means that $(\cdot, R, \cdot) \notin S$ and thus the second winning condition is also met and $\mathcal{B}_1$ wins its game as well. This establishes Equation 2.

Next, we consider $E_\in$, and construct $\mathcal{B}_2$ such that

$$\Pr[E_{\notin}] \leq \mathbf{Adv}^{\text{link}}_{\text{rs},\mathcal{B}_2}(\lambda). \tag{3}$$

Briefly, $\mathcal{B}_2$ uses its given set of public keys as the set of recipient public keys to give to $\mathcal{A}$. When $\mathcal{A}$ queries the AD and AW oracles, $\mathcal{B}_2$ behaves honestly. When $\mathcal{A}$ queries the CORR and HW oracles, $\mathcal{B}_2$ uses its own CORR and SIGN oracles to answer the queries (just as $\mathcal{B}_1$ does). At the end of the game, $\mathcal{B}_2$ extracts the signatures both from the transaction output by $\mathcal{A}$ and from the transaction tx' such that $(j, \cdot, \mathsf{tx}') \in H_w$ (which exists by assumption of $E_\in$). It then outputs these signatures, together with their corresponding messages and the associated ring. Formally, the code for $\mathcal{B}_2$ (omitting the oracles in which it behaves honestly) is as follows:

```
$\underline{\mathcal{B}_2^{\text{Corr,Sign}}(1^\lambda, \mathsf{PK})}$
$H_w \leftarrow \emptyset$
$(\mathsf{tx}, j) \xleftarrow{\$} \mathcal{A}^{\text{Corr,AD,AW,HW}}(1^\lambda, \mathsf{PK})$
identify $\mathsf{tx}'$ such that $(j, \cdot, \mathsf{tx}') \in H_w$
return $(\mathsf{tx}[pk], \mathsf{tx}'[pk], \mathsf{tumblers}[j].\mathsf{keys}_B, \mathsf{tx}[\mathsf{data}], \mathsf{tx}'[\mathsf{data}])$

$\underline{\text{Corr}(\ell)}$
return $\text{Corr}(\ell)$

$\underline{\text{HW}(j, \ell)}$
$R \leftarrow \mathsf{tumblers}[j].\mathsf{keys}_B$
if $(pk_\ell \notin R)$ return $\perp$
$(pk_{\mathsf{ephem}}, sk_{\mathsf{ephem}}) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$
$\sigma \xleftarrow{\$} \text{Sign}(R, \ell, pk_{\mathsf{ephem}})$
$\mathsf{tx} \leftarrow \mathsf{FormTx}(sk_{\mathsf{ephem}}, \mathsf{tumblers}[j], 0, \sigma)$
add $(j, \ell, \mathsf{tx})$ to $H_w$
$\mathsf{ProcessWithdraw}(\mathsf{tumblers}[j], \mathsf{tx})$
return $\mathsf{tx}$
```

By the assumption that $(j, \cdot, \cdot) \in H_w$, we know that $\mathcal{B}_2$ is efficient and such a transaction tx' can be found. By the first winning condition of the theft prevention game, we know that $\mathsf{tumblers}[j].\mathsf{keys}_B \subseteq \mathsf{PK}_B \setminus C$. Given that $\mathcal{B}_2$ uses $R \leftarrow \mathsf{tumblers}[j].\mathsf{keys}_B$ and that the sets C are the same across the two games, this implies that the second winning condition of Definition A.3 is met, as if $R \subseteq \mathsf{PK} \setminus C$ then $|R \cap C| + |R \setminus \mathsf{PK}| = 0$. By the second winning condition of the theft prevention game, we know that $\mathsf{VerifyWithdraw}(\mathsf{tx}, \mathsf{tumblers}[j]) = 1$, and as $(j, \cdot, \mathsf{tx}') \in H_w$ we know that tx' was formed honestly and thus, by correctness, that $\mathsf{VerifyWithdraw}(\mathsf{tx}', \mathsf{tumblers}[j]) = 1$ as well. Looking at the code for $\mathsf{VerifyWithdraw}$ in Figure 1, this further implies both that $\mathsf{RS.Verify}(R, \mathsf{tx}[pk], \mathsf{tx}[\mathsf{data}]) = 1$ and $\mathsf{RS.Verify}(R, \mathsf{tx}'[pk], \mathsf{tx}'[\mathsf{data}]) = 1$ (which means the first winning condition of the linkability game is met) and that $\mathsf{RS.Link}(\sigma, \sigma') = 0$ (which means that the third winning condition is met). This establishes Equation 3, and thus Equation 1 and Lemma 5.4.