Obscuro: A Bitcoin Mixer using Trusted Execution Environments

Muoi Tran[1], Loi Luu[1], Min Suk Kang[1], Iddo Bentov[2], and Prateek Saxena[1]

[1]National University of Singapore
[2]Cornell University

Abstract

Bitcoin provides only pseudo-anonymous transactions, which can be exploited to link payers and payees – defeating the goal of anonymous payments. To thwart such attacks, several Bitcoin mixers have been proposed, with the objective of providing unlinkability between payers and payees. However, existing Bitcoin mixers are not under widespread use and can be regarded as either insecure or inefficient.

We present Obscuro, a highly efficient and secure Bitcoin mixer that utilizes *trusted execution environments* (TEEs). With the TEE's confidentiality and integrity guarantees for code and data, our mixer design ensures the correct mixing operations and the protection of sensitive data (i.e., private keys and mixing logs), ruling out coin theft and de-anonymization attacks by a malicious operator. A TEE-based implementation does not necessarily prevent the manipulation of inputs (e.g., deposit submissions, blockchain feeds, TEE's execution states) to the mixer, hence Obscuro is designed to overcome such limitations: it (1) offers an indirect deposit mechanism to prevent a malicious operator from rejecting benign user deposits; and (2) removes the need for storing any operation states outside of the TEE, thereby denying the possibility of state-rewind in conjunction with eclipse attacks. Obscuro provides several unique anonymity features (e.g., minimum mixing set size guarantee, resistant to dropping user deposits) that are not available in existing centralized and decentralized mixers.

Our prototype of Obscuro is built using Intel SGX, and we demonstrate its effectiveness in the Bitcoin Testnet. Our implementation mixes 1000 inputs in just 6.49 seconds, which vastly outperforms all of the existing decentralized mixers.

1 Introduction

Bitcoin and other cryptocurrencies allow users to transact digital coins without relying on any centralized, trusted party. Although each Bitcoin user can stay *pseudo-anonymous* by generating multiple cryptographic addresses that are used to receive funds, the users' transaction records are publicly available on the blockchain. In other words, Bitcoin's pseudo-anonymity can be regarded as publishing everyone's credit card statements, with the names redacted but the "account numbers" (i.e., cryptographic addresses) and payment amounts visible. The privacy of Bitcoin users can thus be violated by an adversary who is able to link multiple transactions.

This is the authors' full version, last update on December 31, 2018, of a paper that was first posted online on October 5, 2017 and subsequently presented at ACSAC 18, December 37, 2018, San Juan, PR, USA.
https://doi.org/10.1145/3274694.3274750

Indeed, previous works have shown how to deanonymize a large fraction of Bitcoin addresses [55, 47]. For instance, Meiklejohn et al. introduced efficient heuristics for clustering Bitcoin addresses that group all the addresses that are controlled by the same user [47]. This work linked 5000 addresses to users who publicly posted their information on public forums or websites.

Furthermore, without anonymous transactions, the fungibility of the Bitcoin currency is at risk. This is because the coins in circulation could be considered tainted to certain degrees (see [54] for a well-known legal case).

There are several proposals for Bitcoin *mixing services* (or *mixers*) that provide better privacy for Bitcoin users [56, 45, 67, 34, 33]. At a high-level, these protocols are inspired by the mixing networks (a.k.a. *mixnet*), an anonymous digital communication technique proposed by Chaum [24]. Specifically, such mixers randomly *shuffle* several sending and receiving addresses so that it is difficult to identify which sending address is linked to which receiving address, given that all the transacting amounts are the same. A strong mixing algorithm entails that linkability within the *anonymity set* — the set of all possible outputs for a particular input — is hard.

Unfortunately, existing Bitcoin mixers are either known to be *vulnerable* to a number of attacks that can be launched by malicious mixer operators, or *inefficient* due to prohibitive algorithmic/communication overhead for large (e.g. hundreds or thousands) mixing set sizes. In general, centralized Bitcoin mixers require the users to trust the mixer operators [20, 67], and these mixers are susceptible to malicious operators who could steal the users' coins (i.e., scam mixers[1]) or break the anonymity guarantees by leaking the payer-payee address records. Even with a zero-knowledge proof based mixer design which aims to remove the trust in the operator (e.g., TumbleBit [33]), centralized mixers still appear to be vulnerable to some de-anonymization attacks launched by a malicious operator who can selectively accept or reject the users' deposits (see Section 2.3). Alternatively, there are several decentralized Bitcoin mixers that operate with mutual distrust [45, 75, 56, 18]. However, decentralized mixers demonstrate limited scalability (e.g., only mix among 50 peers [45, 56]), suffer from long wait times (e.g., a few hours) for finding other mixing parties [18], or assume a fraction of honest mixing parties (e.g., 2/3 of parties are honest [75]).

Our solution. As a new design point in the space of Bitcoin mixers, we propose a mixing protocol and system called Obscuro. Obscuro is designed to withstand an adversarial operator who aims to steal coins submitted to the mixer, or aims to violate the promised anonymity guarantees (e.g., unlinkable transactions or guaranteed mixing set size).

To the best of our knowledge, Obscuro is the first design that achieves mixing time of a few seconds for anonymity set sizes of thousands of users, while defeating a variety of well-known attacks (see Table 1 for comparison to existing work) in a strong threat model. Further, the Obscuro mixing protocol is much simpler than previous protocols that rely on complex cryptographic operations to achieve similar or weaker security goals. Obscuro is Bitcoin compatible and does not require any changes to the Bitcoin protocol.

Obscuro utilizes the hardware-based *trusted execution environments* (TEE) to protect its mixing operations from potentially malicious software components (including the operating system) of the mixer platform. Obscuro (1) *isolates* its codebase from malicious operators and prevents them from disrupting the mixing operations (e.g., stealing users' deposits); (2) allows users to *verify* the correct implementation of the promised anonymity guarantees (unlinkability, minimum mixing set size) before they submit their coin deposits; and (3) maintains a TEE-based *simple* protocol archi-

[1]This concern is real. There have been several reported cases where legitimate-looking Bitcoin mixing services disappeared with the users' deposits [1, 2].

tecture and thus exhibits extremely efficient mixing operations, so that its mixing set size is limited only by the inherent Bitcoin block size.

OBSCURO has a generic design that is compatible with various trusted execution environment techniques. In this work, we implement OBSCURO using a recent trusted computing capability called Intel SGX [15, 46]. Intel SGX allows applications to be run in a special memory region, called an *enclave*, isolated from all other software on the platform. The content of the enclave can be remotely attested, and is encrypted during runtime when stored in RAM — which provides strong memory integrity and confidentiality.

Challenges. Although trusted execution environments (specifically SGX in our paper) offer strong memory isolation that can enhance the security and efficiency of Bitcoin mixers, malicious mixer operators could still potentially control the entire *worldview* of the enclaved mixer, by manipulating inputs that are sent to the mixer. This creates nontrivial design challenges. First, the malicious mixer operator can selectively prevent benign users from participating in the mixing service, in order to reduce the anonymity set size of a mixing operation. Second, the malicious mixer operator can block or modify the blockchain feeds from the outside world to the enclaved mixer, with fake or stale blocks that may "eclipse" the worldview of the enclave. Worse still, the eclipsed view may then allow de-anonymization by a state-rewind attack [26, 65] on the SGX enclave (cf. Section 2.3).

To overcome these challenges, the OBSCURO design removes all direct network interactions between users and the mixer, by employing an *indirect* deposit method via the blockchain itself. Since the users never communicate with the OBSCURO server and instead interact only with the decentralized Bitcoin network, benign user participation is assured, and IP-level de-anonymization attacks are mitigated. OBSCURO is also carefully designed to be *stateless*, eliminating the adversary's advantage from rewinding the mixer's state to the past. We show that an eclipse attack against OBSCURO reduces to a simple denial-of-service attack by the malicious operator, which is well handled by the OBSCURO's guaranteed refund mechanism.

We implemented a prototype of OBSCURO and run several experiments on both Bitcoin Regtest and Bitcoin Testnet. Our results show that OBSCURO is efficient in mixing a large set of transactions. For example, OBSCURO takes only 6.49 seconds to mix 1000 transactions. We also demonstrate that the SGX-induced performance overhead is negligible (i.e., $3 - 5\%$ of mixing time). This is significant since it simplifies the protocol and removes the need for complex cryptographic operations.

Contributions. This paper makes the following contributions:

- We propose OBSCURO, the first Bitcoin mixer that utilizes hardware-based trusted execution environments. Our solution offers strong security and anonymity guarantees against malicious mixer operators, which have not been achieved by any existing mixers.

- We address a family of powerful attacks that manipulate the worldview of the isolated mixer operations in order to reduce the anonymity set sizes (e.g., the number of benign users in mixing rounds). This is done by avoiding any direct communications between the users and the mixer, and removing the need for state storage for the mixer operations.

- We implement a prototype of OBSCURO with the Intel SGX technology, and demonstrate its effectiveness in terms of mixing times for various sizes of anonymity sets. Our results show that OBSCURO is efficient and ready to be deployed in practice. We also consider various

operational issues and offer some practical suggestions (e.g., who should operate OBSCURO and collect mixing fees).

2 Problem Definition

In this work, we consider the problem of designing a Bitcoin mixer that transfers coins from a set of sending addresses $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$ to a set of receiving addresses $\mathcal{R} = \{r_1, r_2, \ldots, r_n\}$, while providing strong anonymity and security properties in the presence of a wide range of attacks against the mixer. Particularly, a mixer must provide *unlinkability* between $\mathcal{S}$ and $\mathcal{R}$. That is, when the n pairs of addresses are all benign (i.e., belong to honest users), it is hard for an adversary to determine who is sending to whom by inspecting the execution of the mixing protocol and the transactions in the public ledger. Also, a mixer's anonymity set size (i.e., the number of benign address pairs among the n pairs) should not be affected by the adversary. Furthermore, the mixer needs to be secure against a malicious operator of the mixing service, in the case that the operator attempts to disrupt the mixer's service (e.g., by stealing coins).

We begin by explaining why the Bitcoin protocol deficient in terms of unlinkability (Section 2.1), and then define our threat model (Section 2.2). To motivate the challenges of designing a mixer using trusted execution environments, we illustrate a baseline solution and highlight the non-trivial remaining challenges (Section 2.3). We end the section by comparing our proposed mixer to existing solutions (Section 2.4)

2.1 Bitcoin Primer

We first provide a brief background on Bitcoin [51], focusing on the lack of unlinkability in the protocol. Bitcoin allows users to have pseudonymous identities called *Bitcoin addresses*, or simply *addresses*, which are generated from their public keys.[2] Users send coins by creating transactions that specify the recipient's Bitcoin address. The users (miners and network nodes) run the Nakamoto consensus protocol [51, 32, 53] among themselves, in order to maintain a publicly shared ledger that specifies the coins that each user owns. At the high level, a leader is picked periodically from all miners via a random process that requires solving proof-of-work puzzles. The leader proposes a block which includes a set of new transactions to append to the latest state of the ledger. Blocks are cryptographically chained together, hence the term blockchain.

Bitcoin's *linkable* transactions. Bitcoins are stored and spent in transactions. A Bitcoin transaction may include multiple *inputs* and multiple *outputs*. A transaction input is a reference to an unspent output of a previous transaction (i.e., coins that belonged to the sender), that is determined by the previous transaction's hash (a.k.a. TxID) and a specified index of the output field. Each indexed output of a transaction contains the spending amount credit that goes to the receivers. The total input amount is greater or equal to the total output amount (all funds are spent), and the difference (in case of inequality) is paid to a miner as a transaction fee. Since all the Bitcoin transactions are permanently stored on the public blockchain, a transaction can always be linked (i.e., referenced) back to one or more previous transactions by anyone who observes the network. Therefore, in many instances it is easy to associate all the transactions that belong to a specific user, by performing blockchain analysis [55, 47].

[2] In the rest of this paper we consider a Bitcoin address to be an ECDSA public key, which should not be confused with the ElGamal public key ($\mathsf{pubkey}_{\mathsf{mixer}}$) that OBSCURO will use for encryption.

Bitcoin scripts. Bitcoin uses a stack-based scripting language to express the conditions that allow funds to be transferred in the transactions. In standard transactions, the `scriptPubKey` script in each transaction output contains a hardcoded public-key that determines which user can spend the coins in the next transaction, and the `scriptSig` script in each transaction input allows a user to embed a signature (that is verified against the aforementioned public-key) and thereby spend the coins. This means that anyone who knows the correct private key can spend the coins in the next transaction. The scripting system also supports *time locking* via the `OP_CHECKLOCKTIMEVERIFY` opcode [66]. At a high level, using `OP_CHECKLOCKTIMEVERIFY` with a time-lock value (block height or block timestamp) can make a transaction output unspendable until a certain block has been reached. Bitcoin scripts also support `OP_RETURN`, a special opcode that allows adding to the blockchain up to 80 bytes of arbitrary data per transaction [16].

Transaction mixing. There exist several operating Bitcoin mixers in the market, such as Bitcoin Blender [6], Helix by Grams [11], and Bitcoin Fog [8]. These mixers first ask a sender with a Bitcoin address s_i to send x Bitcoin to the centralized mixer address, along with her desired receiving address r_i. Upon receiving n transactions of x bitcoins each from $\{s_1, s_2, \ldots, s_n\}$, the mixer shuffles the transaction input order so that the sending addresses s_i and the receiving addresses r_j are randomly permutated. The mixer then creates and publishes the n permutated output transactions to the Bitcoin blockchain. Therefore, at the end of a mixing round, a user receives coins that are not associated with a particular sender, which achieves unlinkability between her addresses (i.e., s_i and r_i).

2.2 Threat Model

We consider a strong adversary that has compromised the operating system (OS) of the mixer platform, which we refer to as a *compromised OS*. A compromised OS can access any system resource that is under its control (e.g., access any physical memory address). The adversary can also leverage the compromised OS to actively read, modify, or drop any message that should be sent or received by the mixer. In practice, the operator of the mixer platform (e.g., cloud provider) may be malicious, and even if the operator is honest the mixer's OS may be compromised by a remote adversary. We further assume that the malicious OS can collude with a non-negligible fraction of the Bitcoin miners. The fraction of computational power that is controlled by the adversary is assumed to be below half, as otherwise the security of the Bitcoin system itself does not hold.

The compromised OS has several attack goals. It aims to *steal* coins that honest users submitted to the mixer, *deanonymize* a targeted user transaction, or *reduce* the anonymity set size (i.e., the number of benign deposits that are mixed in a mixing round).

2.3 Challenges and Solutions

To protect the mixing operations from a compromised OS and offer strong anonymity and security guarantees, we consider utilizing a hardware-based *trusted execution environment* (TEE). A hardware-based TEE offers significant benefits for our mixer design as it provides the 'isolated execution' (i.e., the mixer is isolated from all other operations in the platform) and 'remote attestation' (i.e., a third party can verify the correctness of the mixer's operations) properties [74]; refer to Section 3.3 for more detailed definitions.

In the following, we present a baseline mixer solution using TEE. In that, we highlight that although the isolation and remote attestation properties are useful, a Bitcoin mixer with a naive

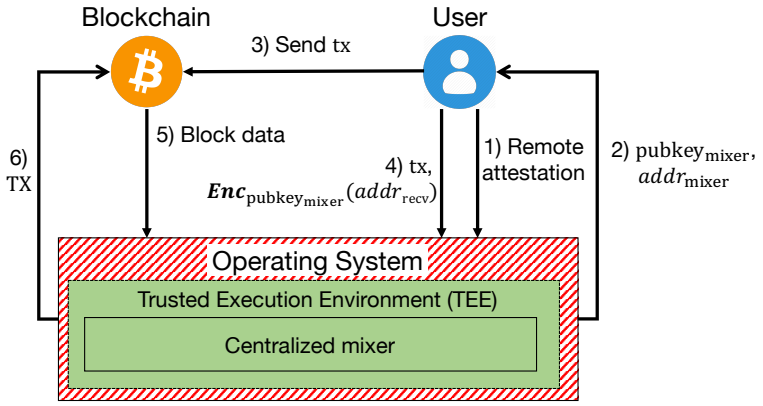


Figure 1: A baseline solution using a hardware-based trusted execution environment (TEE). Components in green background are trusted, while the ones in red stripped background are untrusted.

TEE implementation would fail to address a number of anonymity attacks, and even introduces new vulnerabilities.

A baseline solution using TEE. In this baseline solution, we implement the centralized mixer functions within a hardware protected TEE memory region (or simply TEE), as shown in Figure 1. The mixer within a TEE interacts with its users and the blockchain, i.e., it accepts coin deposits from users, shuffles the deposits, and publishes the mixed transaction to the blockchain.

1) A user requests remote attestation to ensure that the correct mixer implementation has been loaded, and then establishes an authenticated channel with the mixer in the TEE.

2) The user receives the public key ($\mathsf{pubkey}_{\mathsf{mixer}}$) and the Bitcoin address ($addr_{\mathsf{mixer}}$) of the mixer.

3) The user sends coins to $addr_{\mathsf{mixer}}$ by broadcasting a transaction tx to the Bitcoin network.

4) Once the transaction tx is accepted by the blockchain, the user notifies the mixer by submitting tx, together with a returning address $addr_{\mathsf{recv}}$ that is encrypted using the mixer's public key.

5) The mixer retrieves latest blocks from the blockchain and validates that tx is included.

6) The mixer shuffles the returning addresses of all the users who submitted a deposit, and publishes one big transaction TX that has the multiple deposit transactions as inputs and the shuffled returning addresses as outputs. The address permutation is destroyed immediately after each mixing round.

As expected, the strong memory isolation makes it hard for the malicious OS to tamper with the mixer's operations and learn the address permutation. Particularly, this baseline solution effectively addresses the following two attacks:

(1) *Coin theft.* The adversary steals the submitted coins by tricking users to submit coins to the adversary's address, or by compromising the private key of the mixer's Bitcoin address.

(2) *Permutation leaks.* The adversary directly learns the permutation between sending addresses and receiving addresses, by reading the mixing logs.

The remaining anonymity set reduction attacks. Although TEE based security guarantees are useful to address the two attacks above, there exist some remaining challenges. Even with the strong memory isolation properties, the malicious OS can still control the complete *worldview* of the mixer within a TEE, and can tamper with the messages that the TEE sends and receives (e.g., messages 1), 2), 4), 5), and 6) in Figure 1), in order to *reduce* the size of the anonymity set.

Let $\mathcal{S}$ and $\mathcal{R}$ be the sets of benign sending and receiving addresses in a mixing round, respectively. Also, let $\mathcal{S}^A$ and $\mathcal{R}^A$ be the sets of adversarially-generated sending and receiving addresses tumbled together with $\mathcal{S}$ and $\mathcal{R}$ in the same mixing round, respectively. When $\mathcal{S}$ and $\mathcal{S}^A$ are mixed together in the same mixing round, the honest users believe that the anonymity set size of their mixing round is $|\mathcal{S} \cup \mathcal{S}^A|$, as they do not recognize malicious deposits. However, the *effective anonymity set size* of the mixing round is in fact only $|\mathcal{S}|$, since the adversary can de-anonymize the Sybil deposits $\mathcal{S}^A$ beforehand. Hence, if an adversary can reduce the size of benign deposits $|\mathcal{S}|$ while increasing the Sybil deposits $|\mathcal{S}^A|$, she can effectively reduce the anonymity set size without being detected by the honest users.

We present two strategies that reduce the effective anonymity set size: (1) the malicious OS directly drops messages from/to certain users, and (2) the malicious OS forges the blockchain data feeds to the mixer.

(A1) Forced-elimination of benign participants. In the baseline design, a malicious OS can selectively drop the deposits made by honest users, effectively reducing the anonymity set to an arbitrary size. In Figure 1, this can be achieved by dropping the message 1), 2), or 4) during the communication with an honest user. When remote attestation fails due to the dropped message 1), a user would not proceed as the correct mixing operations cannot be ensured. Even if remote attestation succeeds, when the public key and the address of the mixers in message 2) are dropped, the user cannot submit her coins deposit. Moreover, even after a user successfully deposits her coins on the blockchain, if she cannot provide the mixer with her encrypted receiving address $addr_{\mathsf{recv}}$ in message 4), her deposit will not participate in the mixing round. When an adversary reduces the number of benign deposits $|\mathcal{S}|$ by dropping messages from honest users, she can also increase $|\mathcal{S}^A|$ by generating Sybil that would facilitate her de-anonymization attack.

(A2) Blockchain rewinding. In this even stealthier attack, instead of directly limiting the benign users' participation, the malicious OS controls the blocks feed that is sent to the mixer, by providing adversarially-generated Bitcoin blocks to the TEE; in other words, the adversary is eclipsing the mixer from the rest of Bitcoin network [35]. This requires the adversary to possess some significant amount of computational power (e.g., by colluding with some corrupt miners). In the fake blocks, the adversary includes only a few targeted users' transactions, along with many Sybil transactions; i.e., the adversary reduces $|\mathcal{S}|$ and increases $|\mathcal{S}^A|$ in the fake blocks. Once the mixer gets the fake blocks, it performs the shuffling and outputs a mix transaction that corresponds to the reduced anonymity set size. Following that, the adversary *rewinds* the state of the mixer to the point right before she fed the fake block data; specifically, using state-rewind attacks on Intel SGX [65, 43]. The mixer would then performs another mixing round with the real Bitcoin blocks that include all the benign deposits, since it is incapable of detecting the state reversal. However, the adversary has already has broken the anonymity of targeted users in the fake world. Worse still, the adversary may first let the real mix go through on the Bitcoin mainnet, then rewind the state of the mixer and feed it the fake blocks, as this would allow the adversary to generate the PoW for the fake blocks at an arbitrarily slow pace.

	Mixers	Permutation leak prevention	Coin theft prevention	Resistant to dropping participants	Minimum mixing set size guarantee	Join-then-abort resistance	Large anonymity set	Minimum mixing time
Decentralized	CoinJoin [45]	✗[§]	✓	✓	✓	✗	small/medium set	1 block
	CoinShuffle++ [57]	✓	✓	✓	✓	✗	small set*	1 block
	CoinParty [75]	✓	2/3 users honest	✓	✓	✓	✓	2 blocks
	Xim [18]	✓	✓	✓	✗	✓	✓	several blocks (hours)
Centralized	MixCoin [20]	✗	✗ (accountable)[‡]	✗	✗	✓	✓	2 blocks
	BlindCoin [67]	✓	✗ (accountable)[‡]	✗	✗	✓	✓	2 blocks
	TumbleBit [33]	✓	✓	✗	✗ (remix)[†]	✓	✓	2 blocks
	Obscuro	✓	✓	✓	✓	✓	✓	2 blocks

Table 1: A comparison of Bitcoin mixers. SGX specific attacks are not shown as they are only applicable to Obscuro. [‡] Only accountability is provided and thus coin theft cannot be prevented. [§] In CoinJoin, participants can de-anonymize each other's transactions. [†]Users may need to remix repeatedly until desired minimum mixing set size is achieved. *Small anonymity set because of abort attacks and the complexity of the cryptographic protocol.

Scope. In Obscuro, we do not claim to address the inherent limitations of generic mixnet systems: (1) Obscuro deters Sybil attacks by enforcing mixing fees (on top of the transaction fees), as with other cryptocurrency mixers [33, 18, 57], (see the cost analysis of Sybil attacks in Section 4); however, we do *not* prevent cost-insensitive adversaries from launching Sybil attacks that flood the mixer with large numbers of adversarially-generated inputs; (2) Similarly to all the centralized mixing services, Obscuro is susceptible to denial-of-service (DoS) attacks on the mixer's server (though users will never lose their funds, due to a refund mechanism). Decentralized mixers are susceptible to a more severe form of DoS attack, namely join-then-abort by malicious users (cf. Table 1 and Section 8). To mitigate DoS, many independent instances of the Obscuro service can be operated by different reputable servers; and (3) Obscuro does not have special logic for better anonymity across multiple mixing rounds. Thus, an adversary that reduces the anonymity set size in a mixing round might be able to further exploit the data by carrying out an *intersection* attack [23, 29, 40, 48], as in other mixnet based systems.

Our security analysis of Obscuro assumes the existence of a secure underlying TEE technique. Specifically, side-channel attacks against Intel SGX have been actively studied in the past few years [70, 62, 49, 41, 50, 59]. While direct attacks against TEE platforms are beyond the scope of this work, in Appendix A.1 we extend the discussion on side-channel attacks against Intel SGX and the proposed mitigation mechanisms.

Assumptions. We assume that the majority of Bitcoin miners are honest and thus the underlying Bitcoin network is secure. We assume that users can verify the correctness of the Obscuro codebase [13], which contributes an additional 2.4K source lines of code (SLoC) to the trusted computing base; see Section 6.

2.4 Existing Solutions

Existing centralized mixers are susceptible to one or more of the above-mentioned attacks. First, MixCoin [20] and many operating centralized mixers are vulnerable to permutation leak attacks by malicious mixers. Second, all existing centralized mixers are susceptible to Sybil attacks with targeted user selection because they do not prevent a malicious mixer/operator from selectively accepting/dropping mixing requests. Third, some centralized mixers, such as MixCoin [20] or BlindCoin [67], are also vulnerable to coin theft — they provide accountability for their mixing services, but not theft prevention.

There exist several decentralized mixers that can defend against many (or even all) of the aforementioned attacks. However, most decentralized mixers demonstrate limited scalability (e.g., mix with only 50 participants or less [45, 56, 57]), suffer lengthy waiting time (e.g., a few hours) for finding other mixing parties [18], or require a strong assumption regarding the mixing parties (e.g., 2/3 of the parties are honest [75]). This makes their large-scale adoption challenging.

Table 1 summarizes the comparison between the different mixing schemes in terms of their anonymity and security against the attacks we mentioned, as well as other well-known attacks. Note that **(A2)**–Blockchain rewinding is not present in the table as it is only applicable to Obscuro. Detailed analysis of our proposed solution and the related work is given in Section 4 and Section 8, respectively. We also review some public Bitcoin mixers in Appendix A.2, and show that the current mixing services offer poor performance and weak anonymity/security guarantees.

3 Design

In this section, we state the goals of Obscuro and then describe the design of Obscuro in detail, focusing on the two anonymity reduction attacks **(A1)** and **(A2)** that can be launched by the compromised OS.

3.1 Design Goals

Obscuro is designed to offer the following strong security and anonymity properties that defeat the two main attacks.

- **P1.** *Indirect submission of deposits.* Any user who wishes to participate in a mixing round submits a deposit *without direct interaction* with the mixer. Rather, the user will submit the deposit by interacting with the decentralized Bitcoin network. This guarantees that the benign deposits will be included in the mixing round, in the case that a malicious OS attempts to selectively prevent some users from participating in the mix.

- **P2.** *Guaranteed minimum mixing set size.* The mixer guarantees that no less than the minimum number of $\mathtt{N_{min}}$ transactions are shuffled in any mixing round. If the number of submitted deposits in a block is insufficient, the mixer waits for more deposits from the subsequent blocks. When the mixer selects a mixing set for each mixing round, it follows a certain selection policy that has been announced to and verified by the participating users.

- **P3.** *State-rewind resistance.* Obscuro's mixing operation has to resist state rollback attacks that target the mixer's TEE.

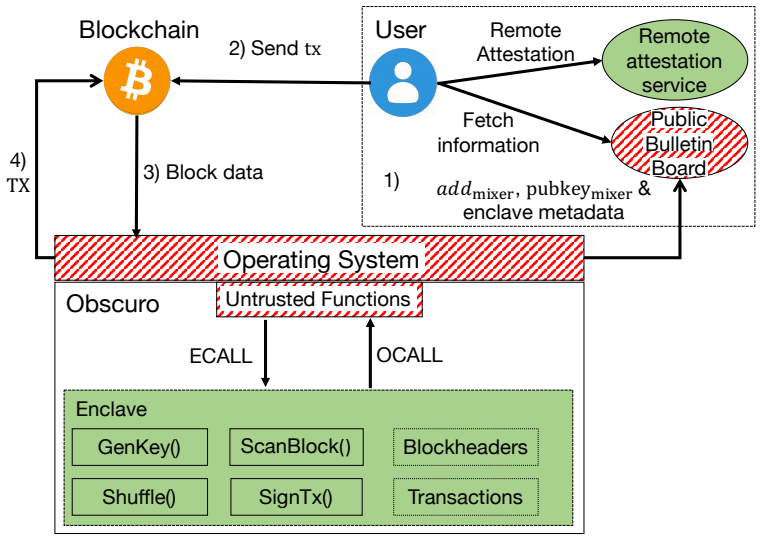


Figure 2: The Obscuro architecture. Components in the green background are trusted, while the ones in the red stripe background are untrusted.

Additionally, Obscuro provides other useful security properties that allow a reliable mixing operation:

- *Guaranteed refund.* For any user who submits a deposit to the mixer, if the deposit is not mixed after a specified waiting time (e.g., when the mixer is under DoS), then the user is guaranteed to have her deposit refunded.
- *Network-layer anonymization.* The mixer cannot learn the IP addresses (or the IP prefixes) of the users during the entire mixing process, and it is also difficult for an external adversary to de-anonymize (cautious) users by inspecting the communication at the IP level.

Obscuro is also designed to be *efficient* and *scalable* for its real-world adoption. That is, its maximum mixing capacity must be large (e.g., several hundreds or thousands of inputs), in fact, it should be limited only by the underlying blockchain's constraints (e.g., maximum transaction size) and not by the mixer's protocol.

3.2 Protocol Overview

Figure 2 illustrates the Obscuro architecture using the Intel SGX capability, showing the communication between the mixer, the Bitcoin *blockchain*, and a *user*. We describe the use of Intel SGX in Section 3.3.

At a high level, Obscuro has four phases: 1) bootstrapping, 2) indirect participation, 3) block scanning, and 4) final announcement. First, in the *bootstrapping* phase, Obscuro randomly generates a fresh public key $\mathsf{pubkey}_{\mathsf{mixer}}$ and a Bitcoin address $addr_{\mathsf{mixer}}$ via `GenKey()`. Obscuro then publishes the new pair ($\mathsf{pubkey}_{\mathsf{mixer}}$, $addr_{\mathsf{mixer}}$) and some metadata to one or more *public bulletin boards*, in order to facilitate the remote attestation. A user, after fetching the information from any of the public bulletin boards, asks a public attestation service to verify whether the correct trusted functions are loaded and being executed in the Obscuro's machine. We discuss the bootstrapping phase in detail in Section 3.4.

Second, if the attestation of OBSCURO was successful, a user sends a deposit transaction tx to the mixer address $addr_{\mathsf{mixer}}$ during the *indirect participation step*. The user attaches her receiving address (which is encrypted with $\mathsf{pubkey}_{\mathsf{mixer}}$) that the mixed coins will be sent to. The transaction tx also ensures the guaranteed refund property of OBSCURO. We extend our discussion on this phase in Section 4.1.

Next, in the *block scanning* phase, OBSCURO connects to the Bitcoin network to download and verify the up-to-date blockchain data, and extract information regarding the users' deposits via `ScanBlock`(). OBSCURO searches for transactions that spent coins to its address $addr_{\mathsf{mixer}}$, and decrypts the user's returning addresses $addr_{\mathsf{recv}}$ from such transactions. OBSCURO verifies the integrity of the transactions of each block by recomputing their Merkle root. OBSCURO also validates each block header according to the Bitcoin consensus rules. The verification procedure begins with a hardcoded block header hash of a recent blockchain checkpoint. To reduce the trust assumption, OBSCURO can also be hardcoded with the current difficulty parameter and the genesis block, and fetch blocks from genesis when the execution starts. Note that it is not necessary to keep all the block data from genesis, since OBSCURO generates a fresh ECDSA keypair in every enclave instance and only extracts deposits that were sent to the address $addr_{\mathsf{mixer}}$ (that is derived from this keypair).

In the *final announcement* phase, OBSCURO waits to receive at least $\mathtt{N}_{\mathsf{min}}$ deposit transactions, and then creates a giant transaction TX that has all the deposits as its inputs and the corresponding return addresses as its outputs. OBSCURO shuffles the outputs and signs TX using the trusted functions `Shuffle`() and `SignTx`(), respectively. Our mixing policy includes several parameters to determine the mixing set for each round: the minimum mixing set size $\mathtt{N}_{\mathsf{min}}$; the maximum capacity of a mixing round $\mathtt{N}_{\mathsf{max}}$; the maximum number of blocks OBSCURO will wait for deposits $\mathtt{B}_{\mathsf{wait}}$; and the number of block confirmations of deposit transactions $\mathtt{B}_{\mathsf{confirm}}$. We elaborate on the mixing policy in Section 5. Lastly, OBSCURO broadcasts the giant transaction TX to the Bitcoin network.

3.3 Use of Intel SGX in Obscuro

Intel's Software Guard Extension (SGX) is a set of security capabilities that offer integrity and confidentiality properties to user-level applications that run on untrusted operating systems or hypervisors [15, 46]. An application implemented in the SGX programming model includes two types of components: (1) trusted components that are loaded and executed inside an SGX *enclave*, a special memory region that is isolated from the untrusted functions including privileged software (e.g., OS), and (2) untrusted components that operate as a non-SGX application outside of the enclave boundary. We summarize the two features of SGX that are particularly useful in our design of OBSCURO.

Enclaved execution. In the SGX model, the untrusted components start an enclave execution by invoking an enclave call (i.e., "ECALL") through well-defined call gates, and get the return value of the enclave function. In many cases where a function running inside the enclave needs to invoke system calls (e.g., disk I/O), it temporarily exits the enclave to outer call (i.e., "OCALL"), an execution in the untrusted space. While the code inside an enclave is able to read/write the application memory outside of the enclave as well as the enclave data in its unencrypted form, the non-enclave code (e.g., the OS) *cannot* access the enclave's memory. Therefore, the trusted part of an SGX application is guaranteed to be executed with confidentiality and integrity.

Remote attestation. Intel SGX supports a *remote attestation* feature, which allows a third party

to be convinced that the correct application code has been loaded in an enclave [15]. In general, a service provider issues an attestation challenge to the enclave application on behalf of a remote user who wishes to attest the application. The enclave then provides a report, which is cryptographically signed by the attestation key of the SGX hardware. The report contains the hash of the enclave's initial contents (i.e., the checksum of the application) and some data computed inside the enclave (e.g., an enclave-generated public key). Next, the attestation report is forwarded by the service provider and verified by the Intel Attestation Service, which is distributed globally [38]. By inspecting the return status, the service provider sees whether the correct application is being executed inside an enclave. In OBSCURO, we remove the reliance on a service provider, and instead distribute the measurement report to some public bulletin boards so that the users can obtain and verify the report independently.

3.4 Bootstrapping Phase

The goal of the bootstrapping phase is to initialize the OBSCURO enclave and allow any user to verify the correctness of the mixer's information before joining a mixing round. Achieving this goal seems easy on the surface but is rather tricky in the presence of the malicious OS that can drop any attestation request of any users. Typically, this phase includes several small steps, as we explain below.

First, when the enclave has been launched and the code is loaded, OBSCURO randomly generates a fresh ECDSA secret key with the corresponding Bitcoin address $addr_{\mathsf{mixer}}$, and a fresh ElGamal secret key with the corresponding public key $\mathsf{pubkey}_{\mathsf{mixer}}$. Since it is crucial to use a reliable randomness source for these keys (see Section 4), we increase the entropy of our random seed in order to reduce the trust in the hardware provider (i.e., Intel in our current implementation). Thus, in addition to the trusted hardware-based randomness provided by the RDRAND to `sgx_read_rand`(), we concatenate extra sources of randomness: OS provided randomness, the SGX trusted clock, and the latest block hash from the Bitcoin blockchain. To predict the random seed that we feed to the key generator, the adversary will need to control all the components that contribute to the seed.

Once $\mathsf{pubkey}_{\mathsf{mixer}}$ and $addr_{\mathsf{mixer}}$ have been generated, the enclave produces some metadata for the remote attestation. This metadata serves as the attestation measurement report from the enclave, which can be used later on to prove that i) the metadata is generated by an SGX enclave; ii) the enclave is running the correct implementation of OBSCURO; and iii) the pair ($\mathsf{pubkey}_{\mathsf{mixer}}$, $addr_{\mathsf{mixer}}$) was generated by the enclave.

Then, the metadata and ($\mathsf{pubkey}_{\mathsf{mixer}}$, $addr_{\mathsf{mixer}}$) are made available on some public bulletin boards. Therefore, new users do not need to interact with the OBSCURO enclave directly to perform the remote attestation procedure before using the mixing service. These public bulletin boards are used for the availability of the data and do not need to be trusted — any tampering with $\mathsf{pubkey}_{\mathsf{mixer}}$, $addr_{\mathsf{mixer}}$, or the metadata will lead to a failed verification by Intel Attestation Service. There are several practical implementations of public bulletin boards, including IPFS [12], data servers with replications, and public blockchains such as Bitcoin itself.

Finally, before using OBSCURO, a user will visit a public bulletin board and fetch the metadata and ($\mathsf{pubkey}_{\mathsf{mixer}}$, $addr_{\mathsf{mixer}}$). The user will then ask the Intel Attestation Service to verify the correctness of the metadata. If the verification passes, the user can proceed to submit her deposit for the mixing round. Note, however, that the interaction with the IAS is *optional* for most of the users since any user or a third party can distribute the attestation validation result from the IAS so that users can verify the attestation report by themselves [36].

Note that $\mathsf{pubkey}_{\mathsf{mixer}}$ and $addr_{\mathsf{mixer}}$ are unique in each instantiation of Obscuro, and live only in the enclave memory; i.e., these values will *not* be stored on disk, and will not be recognized by subsequent instantiations of the enclave. Obscuro uses this *stateless* design to defend against state-rewind attacks on SGX; see Section 4.3 for further discussion.

For better usability, we also add a timestamp to the metadata, and recommend users to join the mixing round only if the enclave's metadata includes a recent timestamp. This is done to reduce the likelihood that users would send a deposit to an address of a defunct enclave, as even though they will get their money back (due to the guaranteed refund mechanism), it is undesirable to have the funds locked for a lengthy period of time.

4 Security Analysis

Here, we show how Obscuro prevents the adversary from blocking honest participations (i.e., attack **A1**). Moreover, we analyze the cost of Sybil attacks against Obscuro to achieve a small effective anonymity set size. We also show that the Obscuro design is not vulnerable to blockchain rewinding (i.e., attack **A2**).

Throughout this section, we assume that the users are not exposed to IP-level attacks. This assumption is reasonable as it is unlikely that an attacker that eavesdrops at the IP-level will be able to link Obscuro transactions, due to the following reasons:

- The indirect deposit method (**P1**) makes it easy for a user to communicate from different locations, since she interacts only with the cryptocurrency network and not with other users.
- Each user transmits only a single message with a deposit and a single message to withdraw her funds (decentralized mixers require the participants to send many more messages), and the user can treat her mix output as "cold storage" and withdraw her funds after an arbitrarily long time period.
- It is possible to use Obscuro to make anonymous payments, e.g., Alice can buy an item from Bob and encrypt Bob's address as her mix output (hence the deposit will request using Alice's IP address and the withdrawal will be requested using Bob's IP address).

Cautious users may also use an IP-level anonymizer (e.g., Tor [30]) for supplementary protection when they deposit their inputs, and when they withdraw their outputs after the mix is done.

To clarify the prospects of a deanonymization attack, we first provide a simple lemma. Let $\mathsf{negl}()$ denote a negligible function, i.e., for every positive polynomial $p(\cdot)$ there exists a sufficiently large n such that $\forall m > n : \ \mathsf{negl}(m) < 1/p(m)$.

Lemma 1. *Suppose that H is the number of honest participants in an* Obscuro *mix. Given an input of an honest participant P, the adversary can guess the output of P in the mix with probability $\frac{1}{H} + \mathsf{negl}(\kappa)$ at the most, where κ is the security parameter of an* Obscuro *instantiation.*

Proof: Since the Obscuro enclave is executing correct code that shuffles all the outputs of the mix by using a secure randomness source, the probability that any specific honest output belongs to P is $\frac{1}{H}$. The adversary controls all outputs except for the H honest outputs, and therefore her guess will be correct with greater than $\frac{1}{H}$ probability only if she breaks the encryption of the outputs that are attached to each deposit. Let $E_1 = \{\text{adversary guessed correctly}\}$ and

$E_2 = \{\text{adversary broke the encryption}\}$. Since the security parameter of the encryption scheme is κ, we obtain

$$\begin{aligned} \Pr(E_1) &= \Pr(E_1 \cap \neg E_2) + \Pr(E_1 \cap E_2) \\ &\leq \Pr(E_1 | \neg E_2) + \Pr(E_2) \leq \frac{1}{H} + \mathsf{negl}(\kappa). \end{aligned}$$

□

Let us note that it is quite plausible to assume that the adversary can break the cryptography with only $\mathsf{negl}(\kappa)$ probability, rather than a less demanding side-channel attack. This is because potential side-channel attacks on ElGamal encryption are an active area of research (with room for improvements, see, e.g., [39]), and the severity of current attacks on the `OpenSSL` implementation of ElGamal (that we use) is relatively low (cf. [72]).

Furthermore, we employ CCA secure variant of ElGamal encryption (cf. Section 5) to prevent deanonymization attacks in which the adversary observes ciphertexts on the public blockchain and uses Obscuro as a decryption oracle. Specifically, if we used (malleable) hashed ElGamal encryption then the adversary could target an honest user's ciphertext $(g^r, \mathsf{hash}(y^r) \oplus m)$ by picking $(g^r, \mathsf{hash}(y^r) \oplus m \oplus \delta)$ as her encrypted return address. At the cost of forfeiting her deposit, the adversary would then deanonymize the user by finding two outputs whose xor is δ.

Given Lemma 1, we now turn to show that H will indeed be large.

4.1 Preventing Selective Dropping of Participants

Obscuro's design prevents a compromised OS from selectively dropping participants, unless they are incautious; that is, the attack **(A1)** is ineffective against prudent users. In Obscuro, a user participates in the mix after (1) getting Obscuro's information (i.e., $\mathsf{pubkey}_{\mathsf{mixer}}$ and $addr_{\mathsf{mixer}}$) and metadata; and (2) using the metadata to verify that Obscuro is running the correct code inside an enclave, and that ($\mathsf{pubkey}_{\mathsf{mixer}}$, $addr_{\mathsf{mixer}}$) were generated by Obscuro's enclave code. For (1), the user fetches the information from public bulletin boards, instead of querying the mixer's server directly. We rely on multiple well-known bulletin boards, and recommend the users to participate in the mix only if they see that the data was uploaded to several of these bulletin boards. Furthermore, each user is advised to upload the data to additional public bulletin boards, in case she sees the data on too few bulletin boards. If the malicious OS prevents Obscuro from publishing the metadata to all of the public bulletin boards, then this would be a straightforward DoS attack, since no user can participate. Once the user fetches the data, she verifies Obscuro's code and the ($\mathsf{pubkey}_{\mathsf{mixer}}$, $addr_{\mathsf{mixer}}$) values by using the trusted remote attestation service — that the malicious OS cannot tamper with.

The user participates in the mix by sending a deposit transaction to $addr_{\mathsf{mixer}}$. Obscuro's design removes any direct interaction with the user, by scanning the transaction from the blockchain directly. This makes it hard and expensive to exclude benign users selectively. The malicious OS can still try to drop users from the mix by conducting an eclipse attack that creates a fake blockchain view to the enclave, but such a strategy would also become just a DoS attack (see Section 4.3).

Another subtle but feasible attack strategy is to influence the miners to include only adversarial deposits in several consecutive blocks. This can be done by either colluding with a large number of miners, or by paying considerably higher transaction fees (relative to normal transactions) to crowd benign deposits out of the blocks. This is plausible because miners are mostly incentive driven, hence they prefer to include transactions with high fees and de-prioritize normal transactions to

subsequent blocks. In Obscuro, we mitigate this risk by rejecting deposits that pay the miners significantly more than the standard fee. Thus, if the adversary attempts to pay much higher fees to have more Sybil transactions in the mix, Obscuro will reject these deposit transactions and waits for other valid deposits. Admittedly, however, Obscuro cannot prevent direct collusion between the adversary and the miners; this risk is inversely correlated to the degree of decentralization in the Bitcoin network.

4.2 Guaranteed Minimum Mixing Set Size

A *minimum* mixing set size provides a lower bound on the quality of the mix. This assures the users that the mixing set size will be large, *before* they send coins to the mixer. To the best of our knowledge, all existing mixers do not provide this guarantee.

As suggested by TumbleBit [33], a user can always inspect the Bitcoin blockchain *after* each mixing round, to see how many other users participated in the mix together with her. If the user is not satisfied with the size of the mixing set, she can *remix* her transaction in one or more subsequent mixing rounds (with additional mixing/transaction fees). This practice is far from ideal, as there is no in-protocol guarantee regarding the mixing set size.

In Obscuro, we leverage the TEE integrity guarantee to enforce a minimum mixing set size $\mathtt{N_{min}}$ (cf. Section 5). This policy can be verified by inspecting the attested enclave code. Let us note that it is desirable to refund the users' deposits – rather than proceed with the mix – when $\mathtt{N_{min}}$ is not reached. The reason is that the honest users pay a considerable mixing fee in order to be secure against Sybil attacks (see Section 4.4), hence they expect a large enough mix in return.

4.3 Eclipse and State-Rewind Attacks

Per **(A2)**–Blockchain rewinding, a malicious OS can carry out an eclipse attack by letting Obscuro communicate only with the attacker's Bitcoin node [35], and feed an adversarially-generated fake blockchain to the Obscuro enclave. This can result in a *fee-free* deanonymization attack because the adversary can pad the fake blocks with Sybil deposits for which she does not need to pay the mixing fee, together with one (or few) deposit from an honest user, and then observe the output of the mixer. Once the adversary de-anonymizes the honest user by using the fake blocks, she rewinds the state of the enclave, and lets Obscuro prepare a new mixing round on the real Bitcoin blockchain with a large set of deposits that includes the already de-anonymized benign deposit.

In any case, Obscuro completely prevents the risk of rewinding the enclave's state, by having a *stateless* design for the Obscuro enclave. Instead of storing some system state (e.g., less than $\mathtt{N_{min}}$ deposits that have been received so far) for permanent operation and recovery from reboots, Obscuro stores no permanent states of the system. Each time that the enclave code is initialized, it uses secure randomness sources (cf. Section 3.4) to generate a unique receiving address. The indirect user deposits are then detected by parsing the blocks that the OS feeds to the enclave, which implies that a set of inputs to a mix in one initialization of the enclave will be disjoint from any other set of inputs in other initializations of the enclave. Hence, in the case that Obscuro is restarted and the enclave code generates a unique receiving address, users who wish to deposit to the new receiving address should encrypt a fresh output address and attach it to their deposit.

Eclipse attacks on Obscuro's stateless enclave are therefore futile as a de-anonymization technique, and such attempts would amount only to a DoS attack on the mixer. To be specific, the resulting giant mix transaction of any mixing operation on the fake blockchain will never be ac-

cepted by the Bitcoin network, because the validity of every transaction is verified according to the Bitcoin consensus protocol (i.e., the giant mix transaction will be rejected because of its invalid Sybil inputs). Honest users will instead get their deposits refunded after the timeout expires, due to the `OP_CHECKLOCKTIMEVERIFY` opcode in the deposit script.

The adversary may attempt to feed a fake chain and a valid chain to the mixer sequentially so that the fake one (which is shorter and used to deanonymize the honest users) becomes naturally abandoned by the Bitcoin consensus run by the stateless Obscuro. However, this attempt can be easily detected by Obscuro checking if the blockchain contains fork and there exists a transaction has been mixed in another fork.

4.4 Cost of Sybil Attacks

Sybil deposits in conjunction with the anonymity reduction attacks **(A1)** and **(A2)** are ineffective in Obscuro. As stated in Section 2.3, Obscuro does not provide protection against cost-insensitive Sybil attacks. Yet, to highlight the impracticality of such attacks, we believe that it is worth analyzing the expected cost of Sybil attacks on Obscuro.

Let H and S be the number of honest and Sybil deposits submitted to Obscuro for the current mixing round, respectively. We also have $\mathtt{N}_{\mathsf{max}}$ as the maximum mixing capacity of a single mixing round; e.g., 430 transactions assuming a standard Bitcoin transaction size of 100 KB (see Section 6). If the total number of mixing requests in this round does not exceed the maximum mixing capacity (i.e., $H + S \leq \mathtt{N}_{\mathsf{max}}$), the effective anonymity set is guaranteed to have size H, due to the indirect deposit submission property (**P1**). However, if the total number of deposits exceeds $\mathtt{N}_{\mathsf{max}}$, some requests must be excluded from the current mixing round. Since the TEE guarantees an unbiased random selection of the mixing requests, the expected anonymity set size as a result of the random selection is $\left(\frac{\mathtt{N}_{max}}{H+S}\right) \times H$, which is inversely proportional to the size of the Sybil requests S. Thus, attacks that aim to generate a small effective anonymity set size may require a prohibitively large number of Sybil deposits S.

For example, consider a conservative scenario in which there are $H = 200$ benign requests and an adversary that aims to reduce the effective anonymity set size to $H = 20$. Since benign deposits cannot be eclipsed, the adversary will need to create 4800 Sybil deposits and thus force the 200 benign deposits to be included in 10 mixing rounds with an average of $H = 20$ per round (when we assume $\mathtt{N}_{\mathsf{max}} = 500$). With 2% mixing fees (where typical mixing fees on the market are in the range of $1\% - 3\%$), this attack costs 96 bitcoins, where the denomination of the mixing round is 1 bitcoin.

5 Implementation

In this section, we describe our Obscuro prototype implementation using Intel SGX. We implement a proof of concept of Obscuro by modifying Bitcoin Core's codebase version v0.13.1 [7], and follow the programming model of Intel SGX applications. Porting the Bitcoin codebase into an SGX enclave is nontrivial since SGX has several limitations on programming general purpose user applications, such as no support for shared object calls in an enclave, and no support for standard system library calls [25]. In our implementation, we utilize a recently proposed framework, called Panoply, that consists of a set of API libraries and compiler extensions that allow enclaves to perform system calls [64]. Moreover, we use the `OpenSSL` library [28], which was already ported

into the enclave by Panoply. Separately, we also port the ECDSA library `libsecp256k1` [69] into the enclave for transaction signing and signature verification. Since `libsecp256k1` operates in constant-time, we thus minimize the likelihood that an adversary will succeed to forge signatures (via a side-channel attack) and steal the users' deposits.

Indirect deposit submission. A user submits her deposit to OBSCURO indirectly, by creating a Pay-To-Script-Hash (P2SH) transaction. The user first chooses a P2SH address as the returning address $addr_{\mathsf{recv}}$ and encrypts it with the $\mathsf{pubkey}_{\mathsf{mixer}}$ of OBSCURO. The public-key encryption scheme for OBSCURO must be *compact* as the ciphertext should be shorter than 80 bytes, the size of `OP_RETURN` field of a Bitcoin transaction. In our implementation, we use the Elliptic Curve Integrated Encryption Scheme (ECIES) over the `secp256k1` elliptic curve. ECIES is CCA secure [17, 22] and its ciphertexts are quite compact for 128 bit security; a ciphertext is a tuple (R, c, d), where R is a 33-byte compressed elliptic curve point, c is the 20-byte encrypted $addr_{\mathsf{recv}}$ with AES counter mode, and d is a 16-byte HMAC tag. With an extra byte as the identifier for fast transaction scanning, a total of 70 bytes are used in the `OP_RETURN` field.

```
OP_IF
<mixer_pubkey> OP_CHECKSIG
OP_ELSE
<time-lock> OP_CHECKLOCKTIMEVERIFY OP_DROP
<user_pubkey> OP_CHECKSIG
OP_ENDIF
```

Figure 3: Structure of the redeem script.

```
Input:
  scriptSig: <user_signature> <user_pubkey>
Output:
  Index: 0
  Value: 0
  scriptPubKey:
    OP_RETURN <identifier> <encrypted_addr_recv>

  Index: 1
  Value: 1000000 %The denomination is 0.01 bitcoin
  scriptPubKey:
    OP_HASH160 <Hash160(redeem_script)> OP_EQUAL
```

Figure 4: Structure of the deposit transaction.

Next, the user constructs a redeem script that follows a predefined format (See Figure 3). This script allows the mixer to spend the deposit, and allows the user to claim back the deposit after the lock-time (if it has not been spent yet). The user then hashes the redeem script, puts the hash in the `scriptPubKey`, and broadcasts the deposit transaction to the Bitcoin network (See Figure 4).

Trusted operations. We describe here the four main trusted functions in OBSCURO, namely a key generation function `GenKey`(), a block scanning function `ScanBlock`(), a random mixing function `Shuffle`() and a transaction signing function `SignTx`(). The `GenKey`() function is used to generate the secret keys for $addr_{\mathsf{mixer}}$ (to be used as `<mixer_pubkey>` in the redeem script) and $\mathsf{pubkey}_{\mathsf{mixer}}$ (that is used by the users to encrypt their $addr_{\mathsf{recv}}$). We use the existing codebase of Bitcoin Core and `libsecp256k1` to generate $addr_{\mathsf{mixer}}$, which is a fresh ECDSA public key. On the other hand,

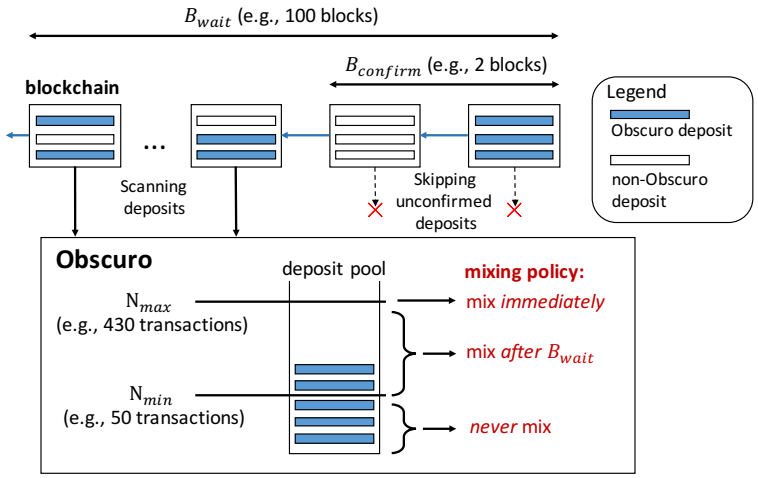


Figure 5: Obscuro's mixing policy.

$\mathsf{pubkey}_{\mathsf{mixer}}$ is an elliptic curve point (i.e., `EC_POINT`) generated using `OpenSSL`. It is worth pointing out that although Obscuro does not change $addr_{\mathsf{mixer}}$ within one execution, the P2SH address used in the deposit transaction is different for each user, since user's public keys (i.e., `<user_pubkey>`) in the redeem script are unique.

The `ScanBlock`() function takes raw Bitcoin block data fetched from the blockchain, and extract all the Obscuro-compliant deposit transactions for future mixing. Upon receiving a block, Obscuro verifies the validity and integrity of the block; specifically, that the block hash meets the difficulty level, and Merkle root hash in the block header matches the transactions in the leaves (this is done by recomputing the Merkle tree). Next, Obscuro scans the transactions of the block and finds all the deposits from users. Due to the 1-byte identifier in the `OP_RETURN` data field, we are able to quickly skip irrelevant P2SH transactions. For each deposit transaction, Obscuro fetches the user's public key, reconstructs the redeem script, and verifies the redeem script's hash. Obscuro then decrypts the 69-byte ciphertext in the deposit's `OP_RETURN` data field, gets the return address, and adds it — altogether with the deposit transaction — to the pool of pending mixing requests.

Based on the mixing policy (see next), Obscuro determines the transaction set for a mixing round. Obscuro crafts a giant transaction TX by putting all the selected transactions as the inputs and the randomly shuffled return addresses as the recipients. We implemented the `Shuffle`() function with the Fisher-Yates shuffle algorithm [19], using the trusted randomness generator. For a transaction TX that has $\mathtt{N}$ outputs, the time complexity of `Shuffle`() is $\mathcal{O}(\mathtt{N})$.

Obscuro invokes the `SignTx`() function to sign the giant transaction TX using the ECDSA signing algorithm of `libsecp256k1`. After all the inputs are signed, Obscuro broadcasts the transaction TX to the Bitcoin network, and finishes the mixing round.

Mixing policy implementation. To specify precisely how the deposits should be collected and when a mixing operation is executed, we introduce three system parameters, $\mathtt{N}_{\mathsf{max}}$, $\mathtt{B}_{\mathsf{wait}}$, and $\mathtt{B}_{\mathsf{confirm}}$, in addition to the minimum mixing set size $\mathtt{N}_{\mathsf{min}}$. In Figure 5, we illustrate how these parameters are used to determine the mixing set for each round. Obscuro continuously monitors the blockchain and collects deposits from the maximum $\mathtt{B}_{\mathsf{wait}}$ blocks (excluding the $\mathtt{B}_{\mathsf{confirm}}$ most recent blocks) since its last mixing operation, to be maintained in the deposit pool.

OBSCURO decides whether to mix the deposits in the pool based on the size of the pool and the number of blocks since the last mixing operation:

- If there are less than $\mathtt{N}_{\mathsf{min}}$ deposits in the pool after scanning $\mathtt{B}_{\mathsf{wait}}$ blocks (since the last mix operation), no mixing is done and the deposits are refunded.
- When the pool reaches $\mathtt{N}_{\mathsf{max}}$ deposits, the mixer immediately starts mixing with the $\mathtt{N}_{\mathsf{max}}$ deposits.
- If there exist at least $\mathtt{N}_{\mathsf{min}}$ deposits after reaching $\mathtt{B}_{\mathsf{wait}}$ blocks since the last mix operation, the mixer performs the mix with all the available deposits.

Note that $\mathtt{B}_{\mathsf{wait}}$ can be set as the time-lock value of the deposits (plus the few $\mathtt{B}_{\mathsf{confirm}}$ blocks for confirmation), so that the user can get refunded immediately in the case that her deposit was not mixed. The purpose of the block confirmation parameter $\mathtt{B}_{\mathsf{confirm}}$ is to avoid a possible invalidation of the giant mix transaction due to chain reversal. The larger that $\mathtt{B}_{\mathsf{confirm}}$ is, the higher the probability that the blockchain becomes irreversible and blocks with users' deposits will not be orphaned. Note that we consider $\mathtt{B}_{\mathsf{confirm}} = 0$ for fair comparison with other mixers in Table 1.

6 Evaluation

Here, we evaluate the bootstrapping, deposit scanning, and mixing operations of the OBSCURO implementation, as well as the on-blockchain practicality. We evaluate OBSCURO on a Dell Latitude E5570 laptop that is SGX-enabled with the 6th Generation Intel® Core™ i7-6820HQ CPU and 8GB of memory. We configure the laptop's BIOS to allocate 128 MB memory for each SGX enclave. We use the Linux 1.6 Open Source Beta version of Intel Software Guard Extensions SDK, Intel SGX Platform Software (PSW), and a driver on Ubuntu Desktop-14.04-LTS 64-bits with Linux kernel version 3.13. OBSCURO is compiled with GCC v4.8 and built for SGX hardware pre-release mode `HW_PRERELEASE` with default optimization flags.

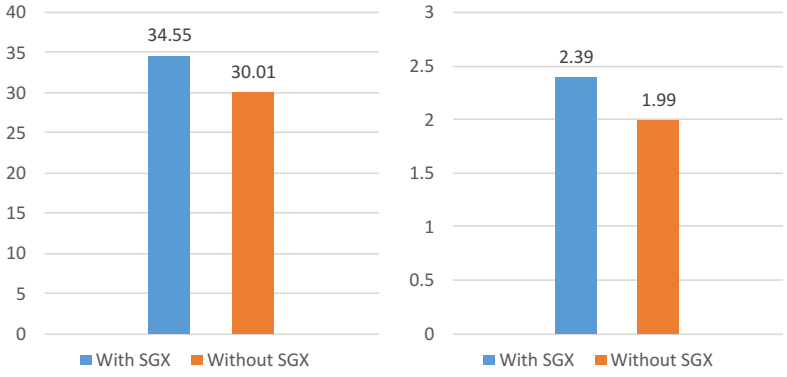
To evaluate the performance of our trusted functions inside the enclave, we test Steps 2), 3), and 4) of the OBSCURO protocol (see Figure 2) in Bitcoin Regression Test Mode (or Regtest). [3] We run each experiment 20 times with two versions of OBSCURO, one in SGX application model and one without SGX support, and the take the average results to measure the added complexity of SGX-related operations.

Bootstrapping Bitcoin blockchain. We measure the time taken to prepare and verify the up-to-date Bitcoin blockchain. This step is done only once per enclave instance (i.e., whenever OBSCURO is launched). As of this writing, the latest block is approximately $200,000$ blocks ahead of the most recent blockchain checkpoint.[4] We measure the time to fetch and verify the block headers of all the blocks after the latest checkpoint. Figure 6a shows that it takes approximately 35 minutes and 30 minutes for the SGX and non-SGX versions, respectively. This is easily acceptable in practice, since the bootstrapping happens only once.

Scanning for Obscuro deposits. We measure the time that it takes to scan Bitcoin blocks and find valid OBSCURO deposits, which also involves ECIES decryption operations. In particular,

[3] The Bitcoin Regtest allows us to set up hundreds of users, generate coins and blocks instantly.

[4] Based on `https://blockchain.info` and `chainparams.cpp` in Bitcoin Core's codebase (September 2017).



(a) Fetching and verifying 200,000 blocks (minutes).

(b) Scanning transactions in 100 blocks (seconds).

Figure 6: Measured time for fetching blocks and scanning transactions with/without SGX.

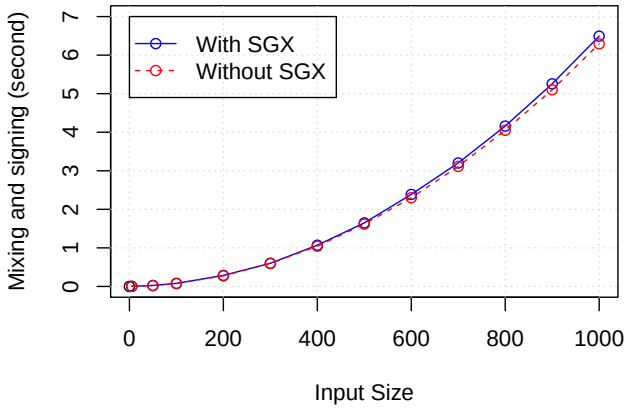


Figure 7: Obscuro signing and mixing time with/without SGX.

we assume a conservative scenario in which Obscuro should scan 100 blocks to search for 2000 Obscuro deposits among a total of 4000 transactions. Figure 6b shows that our implementation scans the deposits very fast (2.39 seconds with SGX, 1.99 seconds without SGX).

Mixing and Signing transactions. We measure the running time of the mixing and signing operations and show that i) Obscuro is scalable and efficient in mixing a large set of transactions; and ii) operating Obscuro with trusted hardware causes negligible overhead. We test the shuffling and transaction signing with different sizes of the mixing set, ranging from 5 to 1000 transactions. Figure 7 shows that the operation time increases as the size of the mixing set increases and SGX programming model causes a very small extra execution time (approximately $3\% - 5\%$). Furthermore, Obscuro can mix one thousand input transactions within seconds (specifically, 1000 inputs in 6.49 seconds). This means that a practical deployment of Obscuro can handle thousands of deposits in a mixing round.

On-blockchain evaluation. Similarly, we deploy Obscuro on Bitcoin Testnet, a global testing environment mimicking the mainnet. We also create multiple users, each with a separate Bitcoin address. Every user sends 0.01 tBTC (testnet bitcoins) to a P2SH address. If the mix is successful, Obscuro sends 0.0095 tBTC to their destination address. The difference between the input and

output amounts is due to the mixing fee. As a result, we successfully mix a large number of input transactions: a standard transaction with 430 inputs[5], and a non-standard transaction with 1000 inputs[6]. The sizes of these two transactions are 100 KB and 228 KB, respectively. Note that the maximum size of a standard transaction is limited to 100 KB, meaning that any larger transaction is considered non-standard. Nevertheless, such transactions can still be mined by paying a higher transaction fee.

Trusted computing base. We measure the size of the trusted computing base (TCB) of our prototype. OBSCURO's trusted functions contribute 1150 source lines of code (SLoC). OBSCURO also requires some changes in the Bitcoin Core implementation which contributes 1292 SLoC. Thus, OBSCURO contributes a total of 2442 SloC to the TCB. The entire TCB includes the Bitcoin Core implementation, two widely used cryptographic libraries (i.e., `libsecp256k1` and `OpenSSL`), and the Panoply implementation [64], which users can audit to verify whether they are deviate from their public codebases [13].

7 Discussion

We discuss some of the operational considerations of OBSCURO; particularly, who should own and operate the OBSCURO instances. Let us first define the *owner* of an OBSCURO's codebase as an individual or an organization that owns/maintains the codebase; e.g., the account holder of a public Github repository of an OBSCURO's codebase. The collected mixing fees are sent to a Bitcoin address under the owner's control, and the address is hardcoded in the OBSCURO codebase. The owner should not be confused with the *operator* of an OBSCURO instance, who controls the platform that hosts the OBSCURO instance (e.g., a cloud provider).

If the owner of OBSCURO is malicious, she may break the anonymity guarantees regardless of the correctness of the mixer operations. This is because imposing mixing fee (as an effective deterrence to Sybil attacks) becomes ineffective when the recipient of the fees is malicious. That is, a malicious owner can generate a large number of Sybil deposits without actually paying any mixing fees, because these fees will be paid to herself. Deciding who should receive the mixing fees is non-trivial, and has been rarely addressed in previous works. We discuss several practical candidates for the recipient of the mixing fees in Section 7.1.

Further, we argue that users should be able to check if an OBSCURO instance is operated by its owner, rather than by a third party. Due to the need for public scrutiny and remote attestation, it is essential for OBSCURO to have an open-source policy, and therefore anyone can launch the same OBSCURO instance in any Intel SGX platform. Although such third party operators will not obtain any financial benefit (as the mixing fees are sent to the owner's address), this may result in DoS attacks. We discuss this problem and propose some operational considerations in Section 7.2.

7.1 Recipient of the Mixing Fees

OBSCURO enforces the mixing fees (e.g., 1–3 percent of deposit value) for each participant and the fees are necessary for Sybil deterrence (see Section 4.4). Note that without mixing fees, rational Sybil attacks become easier as the transaction fees alone may not be sufficient for deterrence. However, if the recipient of the mixing fees is malicious, she can generate a large number of Sybil

[5] https://www.blocktrail.com/tBTC/tx/59e1f4ffe3e6b735f279f340a088597af45f545e6bab4542c82a24d0014b59b9

[6] https://www.blocktrail.com/tBTC/tx/f5230965145ef06eb65595e41ecb701af6c128802a174f34a7b65ac7d44dc9b8

transactions without paying any fees, as the mixing fees will be paid to herself after all. Thus, deciding who should receive the mixing fees is tricky and has been rarely addressed in previous works. We discuss several practical candidates for the recipient of the mixing fees.

The most secure method that guarantees Sybil deterrence is to "burn" the fees, i.e., send the fees to an "unspendable" address (this can be done with a Bitcoin script that always returns `False`).

As a more economically viable solution that is less ideal in terms of security, some reputable charity organizations or privacy advocacy organizations (e.g., EFF [10] or Tor [30]), which are believed to be honest, can be set as the recipient of the mixing fee. It is even possible to allow each user's deposit to choose the recipient of the mixing fee from a list of several reputable organizations (using the identifier byte in the `OP_RETURN` output, cf. Section 5).

7.2 Multiple Obscuro Instances

Launching a new enclave is cheap and thus any third party can spawn an Obscuro instance simply by getting our open-source codebase and running it on its SGX platform. The third party can announce the metadata and keys of its Obscuro instance on some public bulletin boards, just as the real owner of the Obscuro codebase does in the bootstrapping phase (see Section 3.4). Users may thus see many Obscuro instances of the identical codebase that are correctly verified by remote attestation as they are authentic Obscuro code running on SGX platforms. This does not provide any economic gain to the third party operators, since the recipient of the mixing fees (i.e., the real owner's address) is hard-coded in Obscuro's codebase. However, an adversary can launch and announce many Obscuro instances at a very low cost, causing honest users to become confused and deposit their coins to adversary-initiated Obscuro instances — resulting in the shortage of deposits at the owner-initiated Obscuro instance. The adversary can then prevent the actual mixing operation from going through, in the instances that are under her control. Therefore, this amounts to a DoS attack that does not have a substantial cost; e.g., only a handful of SGX-capable Internet-connected machines.

To thwart this threat, the owner can provide a signature that certifies the Obscuro instance that it is operating, along with the metadata that is published in the public bulletin boards. Then, users are advised to deposit coins only to the address of the Obscuro instance that is run by the reputable owner.

As mentioned in Section 2.3, it is useful to have multiple reputable owners who run their Obscuro instances concurrently. If this is the case, then a DoS attack on one server that operates an Obscuro instance will not disrupt the users who participate in a mix that is conducted by another server.

8 Related Work

Several privacy-focused cryptocurrencies have been proposed and built recently [3, 58, 37, 31], but Bitcoin still remains as the most popular cryptocurrency with the largest market capitalization. By contrast, Obscuro to provide a secure and anonymous mixing service for Bitcoin transactions, without any need to modify the current Bitcoin protocol.

Existing mixers can be classified into two main groups based on their design, namely centralized and decentralized ones (see Table 1). In the centralized approach, there typically exists a centralized mixer who receives coins from all the senders, and distributes the coins to all the receivers [20, 67,

34, 33]. Due to its simplicity, this approach is efficient and supports a large number of mixing transactions. However, the centralized mixer becomes a security threat since the mixer may steal users' coins. To sidestep the problem, MixCoin [20] and its successor BlindCoin [67] give users signed certificates to provide accountability. The certificate can hold the malicious mixer accountable for theft by damaging the mixer's reputation. While the accountability prospects are arguable, there is also no guarantee that the mixing server in MixCoin will not violate anonymity; e.g., log the links between the input and output addresses. Our solution guarantees that OBSCURO stores no such information and provides verifiability to users, i.e., the users can validate the remote attestation to see whether OBSCURO is running the intended implementation, before participating.

To tackle the theft and de-anonymization risks in centralized mixers, Blindly Signed Contracts (BSC) combines blind signatures and smart contracts [34]. However, this approach requires protocol changes and thus is not compatible with the current Bitcoin system. TumbleBit, which is inspired by BSC, includes an untrusted intermediate payment hub between the payer and payee, involving a cryptographic puzzle promise and solver protocol among them [33]. Although TumbleBit achieves many desired security properties, it fails to defend against Sybil-based anonymity reduction attacks as it allows a malicious tumbler (or its underlying OS) to selectively drop benign users; see **A1**– Forced-elimination of benign participants in Section 2.3. Moreover, as the tumbler needs to create a time-locked deposit for every escrow in its two-party mixing model, it is vulnerable to a liquidity DoS attack that locks the tumbler's coins for a long period of time (e.g., hours to days). For example, a large number of escrow channels (which would cost some standard fees and mixing fees) for fake receivers can temporarily exhaust all the capital of the tumbler. Some countermeasures exist (e.g., dynamic fee mechanisms, borrowing liquidity from others), but all of them may introduce new vulnerabilities (e.g., allowing the adversary to manipulate the mixing fee structure of the system), or incur an additional operating cost. OBSCURO does not share the same kind of vulnerabilities, since there are no capital constraints on the mixer.

Another line of research focuses on decentralized protocols, in which the participants to communicate among themselves to mix their transactions [56, 57, 45, 18, 75]. Since the final mix transaction requires the approval of all the participants, these protocols are prone to join-then-abort DoS attacks, where the adversary initially participates in the execution of the protocol, but aborts before the end of the execution in order to disrupt the mixing of the honest users. Moreover, the decentralized mixing protocols require a high communication overhead between the participants (specifically, quadratic in the number of participants), which limits their scalability potential. For instance, CoinShuffle [56] and CoinShuffle++ [57] only perform a mix of up to 50 participants. CoinParty [75] relies on an assumption that 2/3 of the peers are honest, which could easily be violated in practice.

Motivated by the need for Sybil- and DoS-resistance, Xim proposes a two-party protocol based on fee advertisement, that the users are required to pay in order to participate [18]. The main drawback of Xim is that the users need to promote themselves several times on the blockchain to find partners, hence the protocol may take hours to terminate.

The trusted hardware, which is available in current commodity CPUs, has opened a new range of research problems — including ones in cryptocurrency research. For instance, a recently proposed *off-chain micropayment channel*, named Teechan, utilizes the trusted execution environment (e.g., Intel SGX) to scale up transaction throughput of Bitcoin transactions [42]. Town Crier relies on Intel SGX to provide authenticated data to the Ethereum smart contracts system [73]. Tesseract uses Intel SGX to build a trust-free cryptocurrency exchange [14].

9 Conclusion

As we are moving towards a digital economy, some sectors of our society already rely on the success of cryptocurrencies. Yet, the fungibility of popular cryptocurrencies — notably Bitcoin — suffered recently, as exchanges and merchants began to consider certain coins in circulation to be tainted. Lack of fungibility harms fair trade and ultimately creates greater centralization pressures, whereas strong anonymity aspects are the key to better fungibility. OBSCURO's efficient operation and strong anonymity guarantees can contribute to the success of Bitcoin and other cryptocurrencies that are exposed to fungibility risks.

10 Acknowledgments

We thank the anonymous reviewers of this paper for their helpful feedback. We also thank Dat Le Tien, Hung Dang, Shweta Shinde and Amrit Kumar for useful feedback on an early version of the paper. This research was partially supported by a grant from Singapore Ministry of Education Academic Research Fund Tier 1 (R-252-000-624-133) and a grant from National Science Foundation (CNS-1617676). This work is also supported in part by research grants to the NUS CRYSTAL Centre.

References

[1] Beware of this scam site: Btcmixer.io. https://www.reddit.com/r/Bitcoin/comments/4rx6fb/beware_of_this_scam_site_btcmixerio/, 2013. Online; accessed Sept 2017.

[2] https://bitcoin-mix.com/ scam exit. https://bitcointalk.org/index.php?topic=1470554, 2013. Online; accessed Sept 2017.

[3] [xmr] monero - a secure, private, untraceable cryptocurrency. https://bitcointalk.org/index.php?topic=583449, 2014. Online; accessed Sept 2017.

[4] How to deanonymize helix mixer transactions. https://www.reddit.com/r/DarkNetMarkets/comments/2ogsgv/how_to_deanonymize_helix_mixer_transactions/, 2015. Online; accessed Sept 2017.

[5] Beginner's guide to bitcoin mixing. https://bitblender.io/guide.html, 2017. Online; accessed Sept 2017.

[6] bitblender.io. https://bitblender.io/, 2017. Online; accessed Sept 2017.

[7] Bitcoin core version 0.13.1. https://bitcoin.org/en/release/v0.13.1, 2017. Online; accessed Sept 2017.

[8] Bitcoin fog. http://foggeddriztrcar2.onion/, 2017. Online; accessed Sept 2017 via Tor.

[9] Bitcoin mixers. https://darknetmarkets.co/category/btc-mixer-tumber/, 2017. Online; accessed Sept 2017.

[10] eff.org. https://www.eff.org/, 2017. Online; accessed Sept 2017.

[11] Helix by grams. http://grams7enufi7jmdl.onion/helix, 2017. Online; accessed Sept 2017 via Tor.

[12] ipfs.io. https://ipfs.io/, 2017. Online; accessed Sept 2017.

[13] Obscuro: A bitcoin mixer using trusted execution environments. https://github.com/bitobscuro/obscuro, 2017. Online; accessed Sept 2017.

[14] Tesseract: Real-time cryptocurrency exchange using trusted hardware. https://www.cs.cornell.edu/~iddo/RTExchSGX.pdf, 2017.

[15] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. Innovative technology for cpu based attestation and sealing. In *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy*, volume 13, 2013.

[16] Massimo Bartoletti and Livio Pompianu. An analysis of bitcoin op_return metadata. *arXiv preprint arXiv:1702.01024*, 2017.

[17] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *J. Cryptology*, 21(4):469–491, 2008.

[18] George Bissias, A Pinar Ozisik, Brian N Levine, and Marc Liberatore. Sybil-resistant mixing for bitcoin. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, pages 149–158. ACM, 2014.

[19] Paul E Black. Fisher-yates shuffle. *Dictionary of algorithms and data structures*, 19, 2005.

[20] Joseph Bonneau, Arvind Narayanan, Andrew Miller, Jeremy Clark, Joshua A Kroll, and Edward W Felten. Mixcoin: Anonymity for bitcoin with accountable mixes. In *International Conference on Financial Cryptography and Data Security*, pages 486–504. Springer, 2014.

[21] Ernie Brickell, Gary Graunke, Michael Neve, and Jean-Pierre Seifert. Software mitigations to hedge aes against cache-based software side channel vulnerabilities. *IACR Cryptology ePrint Archive*, 2006:52, 2006.

[22] D Brown. Standards for efficient cryptography, sec 1: elliptic curve cryptography. *Released Standard Version*, 1, 2009.

[23] Karthekeyan Chandrasekaran, Richard M. Karp, Erick Moreno-Centeno, and Santosh Vempala. Algorithms for implicit hitting set problems. *CoRR*, abs/1102.1472, 2011.

[24] David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[25] Intel Corporation. Intel software guard extensions programming reference. In *Available online: https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf*, 2014.

[26] Victor Costan and Srinivas Devadas. Intel sgx explained. *IACR Cryptology ePrint Archive*, 2016:86, 2016.

[27] Victor Costan, Ilia A Lebedev, and Srinivas Devadas. Sanctum: Minimal Hardware Extensions for Strong Software Isolation. In *USENIX Security Symposium*, pages 857–874, 2016.

[28] Mark Cox, Ralf Engelschall, Stephen Henson, Ben Laurie, et al. The openssl project, 2002.

[29] George Danezis and Andrei Serjantov. *Statistical Disclosure or Intersection Attacks on Anonymity Systems*, pages 293–308. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[30] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In Matt Blaze, editor, *USENIX Security Symposium*, pages 303–320. USENIX, 2004.

[31] Evan Duffield and Daniel Diaz. Dash: A privacy-centric crypto-currency, 2014.

[32] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The Bitcoin backbone protocol: Analysis and applications. In *Eurocrypt*, 2015.

[33] Ethan Heilman, Leen Alshenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg. Tumblebit: An untrusted bitcoin-compatible anonymous payment hub. Technical report, Cryptology ePrint Archive, Report 2016/575, 2016.

[34] Ethan Heilman, Foteini Baldimtsi, and Sharon Goldberg. Blindly signed contracts: Anonymous on-blockchain and off-blockchain bitcoin transactions. pages 43–60, 2016.

[35] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on bitcoins peer-to-peer network. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 129–144, 2015.

[36] Intel. Attestation Service for Intel® Software Guard Extensions: API Documentation. `https://software.intel.com/sites/default/files/managed/7e/3b/ias-api-spec.pdf`, 2017.

[37] Tom Elvis Jedusor. Mimblewimble. `https://download.wpsoftware.net/bitcoin/wizardry/mimblewimble.txt`, 2016.

[38] SP Johnson, VR Scarlata, C Rozas, E Brickell, and F Mckeen. Intel software guard extensions: Epid provisioning and attestation services, 2016.

[39] Eike Kiltz and Krzysztof Pietrzak. Leakage resilient elgamal encryption. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, volume 6477 of *Lecture Notes in Computer Science*, pages 595–612. Springer, 2010.

[40] Amrit Kumar, Clément Fischer, Shruti Tople, and Prateek Saxena. A traceability analysis of monero's blockchain. *IACR Cryptology ePrint Archive*, 2017:338, 2017.

[41] Sangho Lee, Ming-Wei Shih, Prasun Gera, Taesoo Kim, Hyesoon Kim, and Marcus Peinado. Inferring fine-grained control flow inside SGX enclaves with branch shadowing. *arXiv preprint arXiv:1611.06952*, 2016.

[42] Joshua Lind, Ittay Eyal, Peter Pietzuch, and Emin Gün Sirer. Teechan: Payment Channels Using Trusted Execution Environments. In *4th Workshop on Bitcoin and Blockchain Research*, 2017.

[43] Sinisa Matetic, Mansoor Ahmed, Kari Kostiainen, Aritra Dhar, David Sommer, Arthur Gervais, Ari Juels, and Srdjan Capkun. ROTE: Rollback Protection for Trusted Execution. In *Online: `http://eprint.iacr.org/2017/048.pdf`*, 2017.

[44] Mitsuru Matsui and Junko Nakajima. On the power of bitslice implementation on intel core2 processor. In *CHES*, volume 4727, pages 121–134. Springer, 2007.

[45] Gregory Maxwell. Coinjoin: Bitcoin privacy for the real world. In *Post on Bitcoin Forum. Available online: `https://bitcointalk.org/index.php?topic=279249`*, 2013.

[46] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R Savagaonkar. Innovative instructions and software model for isolated execution. In *HASP@ ISCA*, page 10, 2013.

[47] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M Voelker, and Stefan Savage. A fistful of bitcoins: characterizing payments among men with no names. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 127–140. ACM, 2013.

[48] Andrew Miller, Malte Möser, Kevin Lee, and Arvind Narayanan. An empirical analysis of linkability in the monero blockchain. *CoRR*, abs/1704.04299, 2017.

[49] Ming-Wei-Shih, Sangho Lee, Taesoo Kim, and Marcus Peinado. T-SGX: Eradicating Controlled-Channel Attacks Against Enclave Programs. In *NDSS 2017*, 2017.

[50] Ahmad Moghimi, Gorka Irazoqui, and Thomas Eisenbarth. CacheZoom: How SGX amplifies the power of cache attacks. *arXiv preprint arXiv:1703.06986*, 2017.

[51] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *bitcoin.org*, 2009.

[52] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: the case of aes. In *Cryptographers Track at the RSA Conference*, pages 1–20. Springer, 2006.

[53] Rafael Pass, Lior Seeman, and abhi shelat. Analysis of the blockchain protocol in asynchronous networks. In *Eurocrypt*, 2017.

[54] Kenneth Reid. Banknotes and their vindication in eighteenth-century scotland. *David Fox and Wolfgang Ernst (eds), Money in the Western Legal Tradition (Oxford University Press, 2014, Forthcoming); Edinburgh School of Law Research Paper No. 2013/19*, 2013.

[55] Dorit Ron and Adi Shamir. Quantitative analysis of the full bitcoin transaction graph. In *International Conference on Financial Cryptography and Data Security*, pages 6–24. Springer, 2013.

[56] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. Coinshuffle: Practical decentralized coin mixing for bitcoin. In *European Symposium on Research in Computer Security*, pages 345–364. Springer, 2014.

[57] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. P2P Mixing and Unlinkable Bitcoin Transactions. 2017.

[58] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE, 2014.

[59] Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, and Stefan Mangard. Malware Guard Extension: Using SGX to conceal cache attacks. *arXiv preprint arXiv:1702.08719*, 2017.

[60] Jaebaek Seo, Byounyoung Lee, Seongmin Kim, Ming-Wei Shih, Insik Shin, Dongsu Han, and Taesoo Kim. Sgx-shield: Enabling address space layout randomization for sgx programs. In *Proceedings of the 2017 Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA*, 2017.

[61] Ming-Wei Shih, Sangho Lee, Taesoo Kim, and Marcus Peinado. T-sgx: Eradicating controlled-channel attacks against enclave programs. In *Proceedings of the 2017 Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA*, 2017.

[62] Shweta Shinde, Zheng Leong Chua, Viswesh Narayanan, and Prateek Saxena. Preventing page faults from telling your secrets. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, pages 317–328. ACM, 2016.

[63] Shweta Shinde, Zheng Leong Chua, Viswesh Narayanan, and Prateek Saxena. Preventing page faults from telling your secrets. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, pages 317–328. ACM, 2016.

[64] Shweta Shinde, Dat Le Tien, Shruti Tople, and Prateek Saxena. PANOPLY: Low-TCB Linux Applications with SGX Enclaves. In *NDSS*, 2017.

[65] Raoul Strackx and Frank Piessens. Ariadne: A minimal approach to state continuity. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 875–892, Austin, TX, 2016.

[66] Peter Todd. Bip 65: Op checklocktimeverify. *Bitcoin improvement proposal*, 2014.

[67] Luke Valenta and Brendan Rowan. Blindcoin: Blinded, accountable mixes for bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 112–126. Springer, 2015.

[68] Nico Weichbrodt, Anil Kurmus, Peter Pietzuch, and Rüdiger Kapitza. Asyncshock: Exploiting synchronisation bugs in intel sgx enclaves. In *European Symposium on Research in Computer Security*, pages 440–457. Springer, 2016.

[69] P Wuille et al. libsecp256k1: Optimized c library for ec operations on curve secp256k1, 2015.

[70] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 640–656. IEEE, 2015.

[71] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *2015 IEEE Symposium on Security and Privacy*, pages 640–656, 2015.

[72] Yuval Yarom, Daniel Genkin, and Nadia Heninger. Cachebleed: a timing attack on openssl constant-time rsa. *J. Cryptographic Engineering*, 7(2):99–112, 2017. https://ts.data61.csiro.au/projects/TS/cachebleed/.

[73] Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. Town Crier: An authenticated data feed for smart contracts. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 270–282. ACM, 2016.

[74] Fengwei Zhang and Hongwei Zhang. SoK: A Study of Using Hardware-assisted Isolated Execution Environments for Security. In *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016*, HASP 2016, 2016.

[75] Jan Henrik Ziegeldorf, Fred Grossmann, Martin Henze, Nicolas Inden, and Klaus Wehrle. Coinparty: Secure multi-party mixing of bitcoins. In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, pages 75–86. ACM, 2015.

A Appendix

A.1 Intel SGX side channel attacks and mitigation

Although side channel attacks against Intel SGX are beyond our scope (cf. Section 2.3), let us summarize the recent proposed attacks and discuss the countermeasures — which can be directly applied to Obscuro.

Several recent works have inspected the security of Intel SGX based on its use of public resources [26], and uncovered drawbacks in the SGX security model that can lead to side-channel attacks against the enclave. For example, a malicious OS can introduce page faults into the target code, record the accessed pages when page faults occur, and infer the execution flow of the application [71]. This Controlled-Channel attacks can be defeated at the application level [63], or by using the Intel Transactional Synchronization Extensions (TSX) [61].

The adversary can also exploit the race condition among the running threads or synchronicity bugs, and attack the application that runs inside the SGX enclave [68]. This threat can be mitigated by enabling Address space layout randomization (ASLR) protection inside the enclave, as proposed in SGX-Shield [60].

Various cache-timing side-channel attacks have been demonstrated recently, focusing on recovering cryptography keys (of ciphers such as AES [50] and RSA [59]) from running SGX applications. These attacks assume that there exist vulnerabilities in the cryptographic computation inside the enclave (e.g., using *mbedTLS* library to compute RSA signatures [59]). The popular attack technique that is used is *Prime+Probe*, where the adversary evicts a cache set and measures the time that the cache is replaced by the victim's application. This allows the attacker to infers some secrets of the victim, based on the memory accesses. One solution is to use non-vulnerable cryptography libraries that are resistant to cache attacks [21, 44, 52], with some trade-off in performance. Further, the hardware design can be improved to prevent cache-timing and other side-channel attacks [27].

Intel SGX does not clear the branch (i.e., control flow) history when switching between the enclave and non-enclave mode, leaving another side-channel to be exploited. A recently proposed attack, namely branch shadowing, can identify the control flows of the execution inside an enclave [41]. One can mitigate this threat at the application-level by using the Zigzagger compiler, which obfuscates the branch instructions so that the branch executions are oblivious [41].

A.2 Public Bitcoin Mixers

Let us review some popular Bitcoin mixing services currently available in the market, including Bitcoin Blender [6], Helix by Grams [11] and Bitcoin Fog [8]. We observed that a single mixing round of these mixers usually takes anywhere from half an hour to several days (e.g., 102 hours). For instance, Helix mixer requires at least two blocks (i.e., 20 minutes) for deposit confirmation and additional mixing time of 10 minutes to 24 hours. Users of Bitcoin Fog must wait for coin withdrawal from 2 to 102 hours [9]. Bitcoin Blender allows users to customize the coin return time to avoid timing-based blockchain analysis, which usually spans across days (e.g., up to 99 hours) [5].

The centralized mixer operators assert that they delete all the logs from their server after mixing rounds. Yet, there is no guarantee that coin theft and anonymity violation (e.g., secretly logging links between transactions) will not happen. Worse still, all the mixing services that we review do not have any guarantee regarding the anonymity set size. In fact, some de-anonymization attacks have been demonstrated, showing the weak anonymity guarantee of current public Bitcoin mixers [4].