Mixicles:
Simple Private Decentralized Finance

Ari Juels[⋆] Lorenz Breidenbach[∘] Alex Coventry[⋄]
Sergey Nazarov[⋄] Steve Ellis[⋄] Brendan Magauran[⋄]

v1.0
3 September 2019

Abstract

We show how to use oracles to build simple, privacy-preserving decentralized finance (DeFi) instruments we call *Mixicles*.

Mixicles ingest input payments from two or more parties and yield payouts that are conditioned on oracle reports. They are designed to provide privacy for both the terms and outcomes of the financial instruments they execute. At the same time, Mixicles can support rigorous regulatory and auditing requirements.

The most appealing feature of Mixicles is their avoidance of expensive cryptography and complicated contract structures. Mixicles are conceptually simple, and have low on-chain and off-chain resource consumption. We present an example implementation to show how they work and demonstrate their benefits.

1 Introduction

The decentralization of financial markets is well underway, as evidenced by the popularity of blockchain-based digital gold/asset ownership (e.g., Bitcoin) and the adoption of blockchain-based venture capital/crowd sales (ICOs, a.k.a. token sales). Only now, however, are smart contracts beginning to emerge that go beyond basic on-chain ownership through tokenization [59] and tap into other markets. These include the more sophisticated and much larger groups of contracts found in the capital markets as

[⋆]Ari Juels is a faculty member at the Jacobs Institute at Cornell Tech. He co-authored this work in his separate capacity as a technical advisor to SmartContract Chainlink Ltd., in which he has a financial interest.

[∘]Lorenz Breidenbach is an SNF BRIDGE fellow at ETH Zürich. He co-authored this work in his separate capacity as a technical advisor to SmartContract Chainlink Ltd.

[⋄]Smartcontract Chainlink Ltd.

well as the world's largest market by notional outstanding value, derivatives [7]. The aggregate value of capital markets alone (very roughly \$200Tn) [75] dwarfs that of markets that lend themselves to basic tokenization, such as venture capital (approximately \$250Bn in 2018 [53]) and gold (roughly \$9Tn total value at current market price [3, 88]). There is potentially vast untapped value waiting to migrate on chain.

Migrating these more sophisticated classes of financial instruments on chain, though, requires solutions to important technical problems. Two categories of capabilities are firm requirements for these instruments, but absent today from almost all on-chain environments:

1. *Secure and Reliable Oracles:* Smart contracts need access to a robust ecosystem of oracles that provide secure and reliable access to critical external systems like data feeds. Solving this problem is the essence of Chainlink's ongoing mission. Nearly all financial instruments of any complexity need a solution. Something as simple as exercising a stock option, for example, requires knowledge of the current market price of the target equity. There is also clear precedent for this requirement in the over \$200Tn financial contracts today tied to LIBOR and thus reliant on LIBOR data feeds [75].

2. *Smart-Contract Confidentiality*: Smart contracts naïvely implementing financial instruments on blockchains (permissionless or permissioned) can leak critical business intelligence: Monetary amounts, instrument terms, and participating counterparties. In today's complex capital and derivatives markets, privacy is both a given and a legal requirement of most contracts. The reasons for privacy are varied, but often involve protecting against adversarial trading or undesirable market events resulting from specific contract terms becoming public. Meaningful confidentiality in on-chain financial instruments must extend to transactions, oracle queries, and oracle reports.[1]

In this paper, we present our early work on an oracle-based approach to the problem of confidentiality in on-chain financial (DeFi) instruments. Our solution, which we call a *Mixicle*—"*mixer*" + "*oracle*," is a special form of oracle-enabled private smart contract.[2]

Mixicles are inspired by *mixers* or *tumblers*, and have conceptually helpful structural similarities that we use to explain our constructions. But they differ in a few key ways. First, Mixicles are designed for the purpose of preserving privacy for financial instruments, not the concealment of cryptocurrency movements. Second, they rely on oracle reports to determine payouts, rather than random shuffling. Finally, they

[1]One important but subtle issue is worth highlighting: If oracle reports are public, they can be exploited by *parasitic contracts*. These are contracts that reuse freshly delivered oracle data. Parasitic contracts can give rise to complex systemic risks, as we discuss below, so countermeasures are desirable.

[2]We change the 'a' of "oracle" to an 'i' for euphony and visual appeal.

are amenable to audit and regulation. In what follows, we explain what a Mixicle is, how it can be used to implement financial instruments, and how it might be designed.

Our Mixicle design goals are *simplicity* and *efficiency*, as well as *compatibility* with the Chainlink network and its accountability features, i.e., ability to monitor and enforce correct oracle behavior.

The value of simplicity in particular should not be overlooked. Its benefits include ease of implementation and ease of inspection, and thus enhanced security. For this reason, Mixicles exclude heavyweight cryptographic tools such as complex zero-knowledge (ZK) proofs. While elegant and powerful, generalized ZK proofs entail considerable implementation complexity, often relying on customized (virtual) circuitry and state-of-the-art optimizations that few have the expertise to review. The result is a de facto and sometimes dangerous centralization of trust within a small nucleus of experts.[3]

We also avoid the use of trusted execution environments (TEEs) in Mixicles. TEEs can deliver strong confidentiality and high performance, but introduce trust assumptions some users prefer to avoid given recently demonstrated vulnerabilities. We still believe that TEEs should play an important role in blockchain systems, and briefly discuss them in Appendix C.

Following the common terminology in cryptography, we refer to the parties in a financial instrument as *players*. Our focus in this work is on *two-player* Mixicles. Financial instruments with two players are the common case, and two-player Mixicles turn out to be especially efficient and simple to construct. The techniques we introduce, though, can be extended to larger numbers of players. Additionally, our emphasis here is on Mixicle implementation in Ethereum [21, 89], but similar constructions are applicable to other smart contract systems, both permissioned and permissionless, including Hyperledger Fabric [6], Tezos [37], NEO [1], Libra [54], etc.

Our presentation of Mixicles here is preliminary and informal. We don't include rigorous specifications such as ideal functionalities, nor do we give proofs of security. Our goal is to present a useful and flexible framework that may be explored in greater depth for specific applications.

[3]Today, the only meaningful, non-trivial, production use of general-purpose ZK proofs—ZK-SNARKs in particular—is, to the best of our knowledge, in Zcash [43, 72]. Despite the considerable expertise and care of the Zcash team, a long-hidden flaw was identified that could in principle have permitted arbitrary, stealthy minting of coins [39]. The confidentiality properties of ZK mean that the community cannot ascertain whether or not the bug was exploited. Additionally, usability limitations have meant that shielded (ZK) transactions offer limited privacy in practice [49].

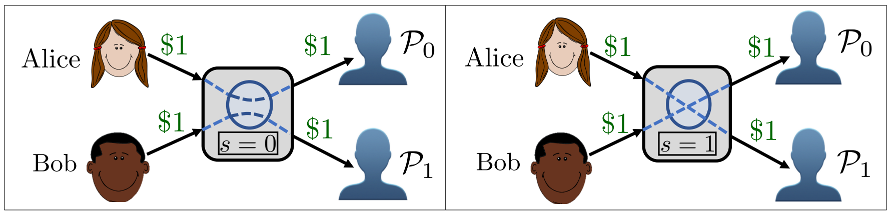


Figure 1: Alice and Bob use a tumbler to pay \$1 each to pseudonymous users $\mathcal{P}_0$ and $\mathcal{P}_1$. Private bit s indicates the confidential payer-payee relationship. If $s = 0$, Alice designated $\mathcal{P}_0$ as payee, and Bob, $\mathcal{P}_1$; vice versa for $s = 1$.

2 Mixicles: Key ideas

2.1 What's a mixer / tumbler?

Mixing is a longstanding approach to privacy in computing systems dating back to the 1980s [23]. A *mixer* or *tumbler* is a basic privacy-enhancing tool used today in a number of cryptocurrencies. Prominent examples include numerous mixing services for Bitcoin, e.g., [60, 64, 78], and mixing via ring signatures as a native feature of Monero [68].

The goal of a tumbler is to improve transaction-layer privacy in systems where *payments are visible on chain*—i.e., nearly every blockchain, public or private.[4] Mixicles are inspired by mixers/tumblers, but do not aim at tumbler-like concealment of funds. Instead, the key observations in Mixicles are that tumblers can be triggered by oracle inputs, rather than randomized, and that they can be made compatible with auditing and regulation.

A tumbler takes payments from one set of users as input, and makes payments to another (possibly overlapping) set of users as output. Its key feature is that it *makes private the correspondence between inputs and outputs*. In other words, as the outside observer of a tumbler, you can't tell who's paid whom.

Fig. 1, for example, depicts a simple tumbler with two inputs and two outputs. Alice and Bob use the mixer to pay two pseudonymous users called $\mathcal{P}_0$ and $\mathcal{P}_1$.

Any observer can see the payments into and out of the tumbler happen on chain. The confidentiality provided by the tumbler, though, means that an observer can't tell whether Alice has designated $\mathcal{P}_0$ as a payee—and Bob, $\mathcal{P}_1$—or vice versa. Inputs are *pooled*, and outputs just draw on the pool. One can think of the tumbler as a payment switch with a private bit s: $s = 0$ means Alice pays $\mathcal{P}_0$ and $s = 1$ means she pays $\mathcal{P}_1$—and vice versa for Bob.

$\mathcal{P}_0$ and $\mathcal{P}_1$ might be users other than Alice and Bob, say Carol and Darlene. Alternatively, they might be pseudonyms representing Alice and Bob themselves.

[4]Like any privacy-enhancing technology, mixers can be used for good, namely to protect user privacy, or ill; see, e.g., [41].

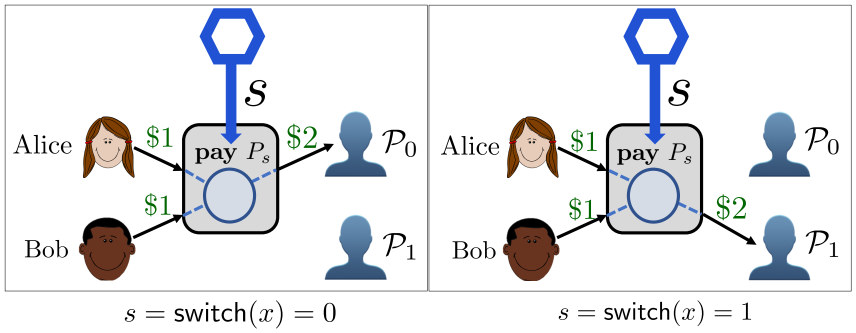


Figure 2: A two-player, binary-option Mixicle into which Alice and Bob each pay \$1. The Mixicle pays one of the players, $\mathcal{P}_0$ or $\mathcal{P}_1$, depending on the value of $\mathsf{switch}(x)$, a predicate on oracle report x.

Why would Alice and Bob pay themselves? To prevent observers from following their transaction history. For example, if, after mixing, $\mathcal{P}_0$ later pays Carol, an observer can't tell: Is it Alice paying Carol, or is it Bob?

Tumblers can, of course, involve more than two users. The more users, the harder it becomes to trace payments going through the mixer. The bigger the crowd a user is a part of, the greater the anonymity she enjoys.

2.2 Mixicles: Key ideas

A standard tumbler's job is to decorrelate sender and receiver identities as in our example above. But the idea might be extended to a more general notion of decorrelating inputs and outputs. In this generalized (if unorthodox) view, one can think of a tumbler as a *payment switch whose internal operation is private.* In a tumbler, this switch is randomized. In a Mixicle, the switch instead takes input from an oracle.

Consider, for instance, a Mixicle that pays all of the money it receives to just one player, choosing this player based on the outcome x of some event, as in Fig. 2.

If $\mathcal{P}_0$ and $\mathcal{P}_1$ are pseudonyms for Alice and Bob, i.e., they're paying themselves, then this Mixicle is a binary bet. Let X denote the space of possible event outcomes, i.e., $x \in X$. Then x might be the result of an election, e.g., $x =$"Democrats" or "Republicans." In this case, if $\mathsf{switch}(x) = 0$ for $x =$"Democrats," then $\mathcal{P}_0$ wins if the Democrats win, and loses if the Republicans win. Alternatively, x might be the result of a horse race or the temperature in Topeka, Kansas, or a Supreme Court ruling, or anything else someone out there might want to bet on.

Of course, financial instruments in general are bets (and bets are financial instruments). X could equally well be the possible states of an asset, e.g., the price of a particular stock. A binary Mixicle would then be a financial instrument known as a *binary option.* We give some examples of binary options and their associated switches in Table 1.

Binary option description	$x =$	$\mathsf{switch}(x)$
Tesla, Inc. stock price above \$1000 at time T?	Ticker TSLA price at time T	$\begin{cases} 0 \text{ if } x \text{ above \$1,000} \\ 1 \text{ if } x \text{ above \$1,000} \end{cases}$
Flippening at time T?	(ETH cap, BTC cap) at time T	$\begin{cases} 0 \text{ if BTC cap} > \text{ETH cap} \\ 1 \text{ if ETH cap} \geq \text{BTC cap} \end{cases}$
Peak NYC temp. in 2025 exceeds 120°F?	Peak NYC temp. in 2025 in F	$\begin{cases} 0 \text{ if } x \leq 120 \\ 1 \text{ if } x > 120 \end{cases}$
U.S. women's foil fencing team wins gold in 2024 Olympics?	U.S. women's foil team's medal status in 2024 Olympics ($\in$ {gold, silver, bronze, no medal})	$\begin{cases} 0 \text{ if } x = \text{gold} \\ 1 \text{ if } x \neq \text{gold} \end{cases}$

Table 1: Some example binary options with switches.

Our example Mixicle in Fig. 2 provides not just confidentiality for the receiver identity, but also critically *makes private the nature and outcome of the underlying bet*, as required for a Mixicle. That is, only Alice, Bob, and the oracle learn X and switch. The payments themselves are visible on chain. If $\mathcal{P}_0$ and $\mathcal{P}_1$ are good pseudonyms, though, i.e., not linkable to Alice and Bob, then the identity of the payee (and loser) of the bet will also remain confidential to observers of the blockchain and even the oracle. (They remain auditable, however, as we explain later.)

For simplicity, we focus in this paper on two-player Mixicles with binary bets, but of course generalization to multiple players and arbitrary switch logic, e.g., non-binary x and switch, is possible.

3 Adversarial Model and Security Requirements

To date, the literature on smart contract security focuses mainly on bug detection and/or formal verification, e.g., [11, 16, 42, 57], or the design of smart contracts for crime or protocol subversion [47, 62]. Some constructions, e.g., [25, 52, 91], also aim at ensuring confidentiality, that is, privacy for contract data with respect to observers and players. Few of these works, e.g., [47], consider oracles.

Mixicles aim to achieve correctness and confidentiality in senses we make precise below. They operate, however, in a setting more complex than that considered by previous works because of the inclusion of *potentially adversarial oracles.* Instead of the usual two sets of players—those internal and external to a contract—the adversarial model for a Mixicle encompasses three types of players potentially interacting with a smart contract $\mathcal{M}$:

- *Oracles*: The oracle $\mathcal{O}$ (or set of oracles $\mathcal{O} = \{O_1, \ldots, O_m\}$) providing data to $\mathcal{M}$.
- *Internal players*: The players participating in the Mixicle, e.g., the Alice and Bob of our examples above.
- *External players*: Other on-chain entities in the system.

Where clear from context, we sometimes refer to "players," without indicating whether they are internal or external.

The goal of a Mixicle is to achieve three key security properties:

- *Confidentiality*: Privacy for the terms—the query, amounts, timing, etc.—of $\mathcal{M}$ as much as possible from external players and $\mathcal{O}$.
- *Auditability*: Mixicles enforce confidentiality as described, but are designed to facilitate transparency to third parties where appropriate.
- *Robustness to oracle faults*: An oracle has wide latitude to misbehave, potentially issuing false or delayed reports, colluding with internal and/or external players, etc. A Mixicle construction should safeguard as much as possible against these behaviors, even if it cannot guarantee correctness.

We now describe these two security properties in greater detail.

3.1 Confidentiality

Mixicle confidentiality has two components: *payment confidentiality* and *query confidentiality.*

- *Payment confidentiality* is privacy for payment information in two senses:
  - *Payment-destination confidentiality:* Privacy for the identities of players to which payments are routed. That is, a decoupling of pseudonyms receiving payouts from input identities of Mixicle players. Ordinary tumblers aim to achieve such confidentiality.
  - *Payment-amount confidentiality:* Privacy for the *amounts* involved in its underlying financial instrument. Achieving such confidentiality in a smart contract system where payment flows are public is challenging. In our Mixicle constructions, we show how to achieve various types of (partial) payment-amount confidentiality.
- *Query confidentiality:* Making switch and x private.

Ideally, confidentiality can be enforced in a Mixicle with respect to *both* external players and the oracle $\mathcal{O}$, something our constructions achieve for payment confidentiality.

Query confidentiality with respect to $\mathcal{O}$ is trickier. As $\mathcal{O}$ itself is evaluating switch, it would seem infeasible to keep query data private from it. Ideally, we would like to achieve a property we call:

- *Strong query confidentiality:* Privacy for switch and x with respect to $\mathcal{O}$.

Such confidentiality *is* in fact possible, using trusted hardware, as in the Town Crier system [91]. It is also achievable using cryptographic techniques newly introduced by Zhang et al. in a system called Deco [93]. Deco enables a user to retrieve content from a TLS-enabled server and prove statements about it in zero knowledge to an oracle, e.g., retrieve a stock price and prove that it has the same ticker a greater price than a committed (ticker, price) pair in $\mathcal{M}$.[5] We omit further details, and do not consider strong query confidentiality in our constructions in this paper. We do, however, consider it an attractive and practical potential enhancement for Mixicles.

An additional form of confidentiality to consider in Mixicle designs is *timing confidentiality.* The time at which a query is resolved—and thus the time $\mathcal{O}$ might trigger a Mixicle—serves as a side channel that can reveal information about switch. We discuss this subtle form of confidentiality in Appendix B.1.

Query confidentiality and freeloading. Query confidentiality has implications beyond privacy. It also provides an additional, subtle security property, namely protection against various forms of *oracle freeloading* [33]. A problem with on-chain delivery of oracle data is that the data can be read and used by *parasitic contracts*—using, e.g., ProvEth [17] or a less expensive oracle query that replays on-chain data. Two problems result: (1) Reuse of oracle outputs is *unfair*, in the sense that while one entity pays for queries, others benefit without compensating the oracle and (2) Reuse of oracle outputs *creates complex dependencies* that can result in hard-to-evaluate and possibly systemic risks of oracle failure. Query confidentiality prevents such abuses.

3.2 Auditability

Confidentiality in Mixicles is enforced strongly with respect to external players and as strongly as possible with respect to $\mathcal{O}$. It is not unconditional, however. There are many reasons why third parties may require access to the internals of a Mixicle, including audits of participating players, regulation of players or regulatory oversight of the ecosystem with an eye to systemic risks, and investigations by law enforcement.

Thus Mixicles are designed also to support what we broadly call:

[5]Deco is somewhat like TLSNotary [2], but supports a much richer range of privacy-preserving proofs, such as those needed for Mixicles. It also supports TLS 1.2 and 1.3, which TLSNotary doesn't.

- *Auditability:* A Mixicle includes an immutable transaction record. A third party such as an auditor can therefore verify player claims about the terms, execution, and payouts of the Mixicle. One option for Mixicles is to require ciphertexts on chain that can be decrypted by an auditor or regulator or by law enforcement. We discuss auditing in Section 5.5.

3.3 Robustness to oracle faults

A fully trusted oracle $\mathcal{O}$ would always comply correctly with the terms of service pre-agreed-upon with internal players, which we denote by terms. In our constructions, terms includes a specification of the predicate switch to be evaluated by $\mathcal{O}$ (with a specification of X, if there's ambiguity) a deadline t_{report} by which $\mathcal{O}$ must deliver a report, i.e., its evaluation of $\mathsf{switch}(x)$, and the delivery target for the report, namely the contract $\mathcal{M}$.

Achieving oracle trustworthiness in practice requires certain assumptions, such as the trustworthiness of trusted execution environments [91] or that a distributed oracle $\mathcal{O} = \{O_1, O_2, \ldots, O_m\}$ has a majority of honest nodes [33]. Ideally, however, an oracle-dependent system, such as a Mixicle, should fail gracefully when these assumptions do not hold.

It is also useful therefore to consider a model addressing key security requirements even if $\mathcal{O}$ misbehaves. In our constructions, we show how to enforce:

- *Accountability:* One helpful side effect of Mixicle auditability is that if $\mathcal{O}$ does not comply with terms, i.e., behaves incorrectly or fails, then any internal player can prove this non-compliance to an auditor (or other third party).

- *Collusion-resistance:* No coalition including $\mathcal{O}$ and players (internal and/or external), can defraud an internal player in the sense of stealing money—except by means of an auditable oracle failure, i.e., $\mathcal{O}$ sending an incorrect report or not sending a report by the designated deadline.

- *Denial-of-service resistance:* If $\mathcal{O}$ fails to provide timely Mixicle input, players can still withdraw their funds.

We aim in our constructions to achieve these properties given arbitrary, i.e., potentially Byzantine, oracle behavior. While Mixicle designers may be willing to make assumptions about the trustworthiness of $\mathcal{O}$, our Mixicle designs offer backstop properties should trust assumptions about $\mathcal{O}$ fail.

3.4 Model limitations

Mixicles do not provide confidentiality, of course, beyond their scope of execution. Any desired confidentiality for the funds coming from *input addresses*—the addresses from which Alice and Bob send payments in our examples—and, more importantly, for funds withdrawn by *output addresses*—those for $\mathcal{P}_0$ and $\mathcal{P}_1$ in our examples—must be addressed through some external means. Players with repeated interactions over an extended period of time, e.g., financial institutions, can benefit from privacy for intermediate transactions in SIMI-PPS and the possibility of making periodic partial withdrawals. We discuss the challenge of withdrawal confidentiality in Appendix A.

Additionally, Mixicles enable users to *prove* oracle malfeasance, i.e., violation of terms through incorrect reporting, to third parties, such as auditors. But Mixicles do not *prevent* such problems. If an instrument has high enough value and $\mathcal{O}$ does not incur substantial penalties for misbehavior, a relying Mixicle will be at risk.

We also omit payments to oracles and the relevant security and design considerations from our model and protocol specifications. (We do, however, incorporate oracle payments into our example Mixicle implementation.)

4 Warm-Up Constructions

In this section, we present two simple, practical Mixicle constructions. We begin with a construction called SIMI ("simple Mixicle") that generalizes the scheme of Fig. 2. We then present a further generalization called *payout slates* that enables privacy for Mixicle payment amounts to players. We realize payout slates in a scheme called SIMI-PS ("simple Mixicle with payout slates").

These two simple constructions convey the basic technical ideas behind Mixicles. They pave the way for presentation in the next section of our main construction, SIMI-PPS ("Simple Mixicle with private payout slates"), which offers more functionality and stronger confidentiality.

4.1 SIMI: Simple Mixicle

Our first, very basic (yet practical!) construction, SIMI, is a two-player, binary-option construction.

We assume that Alice, Bob, and $\mathcal{O}$ can communicate over secure private channels.[6] This is necessary, in fact, as they must agree privately on the terms of the underlying financial instrument in which they are involved.

Our constructions make use of pseudonyms of the form $\mathcal{P}_i$, each of which has an associated (randomly selected) address. In a slight abuse of notation, we let $\mathcal{P}_i$ denote either the pseudonym or its associated address, depending on the context. Similarly

[6]It's possible, of course, to implement such a channel straightforwardly on chain by encrypting messages under players' public keys. Such on-chain communication has the benefit of enabling players to find and interact anonymous with one another exclusively on chain. As it's somewhat expensive, though, and has unclear regulatory implications, we assume an off-chain channel in this paper and in our example Mixicle implementation.

we sometimes conflate $\mathcal{P}_i$ with the player that controls the pseudonym, i.e., either Alice or Bob.

We let $\sigma_{\mathcal{P}}[V]$ denote a signature by player $\mathcal{P}$ (or $\mathcal{O}$) on a value or set of values V. Where clear from context, we drop the subscript.

We use the symbol \$ to denote a monetary amounts (as in the examples above), and additionally render text in green to denote funds in flight in our protocol specifications and figures—as opposed to specifications of monetary amounts.

4.1.1 Basic SIMI construction

SIMI, again, is a generalization of Fig. 2. The main additions are player agreement with $\mathcal{O}$ on the terms of the contract and a timeout that pays out refunds if a fault occurs. SIMI includes three phases:

Setup. In an *off-chain* setup phase, Alice and Bob first generate respective public / private key pairs. They *randomly* assign these addresses to pseudonyms $\mathcal{P}_0$ and $\mathcal{P}_1$. In other words, they pick a random bit b; Alice assigns an address under her control to $\mathcal{P}_b$, and Bob similarly to $\mathcal{P}_{1-b}$. They thereby make private the association between output addresses and input addresses.

The players also agree on a specification S of the switch switch to be used in the Mixicle and the time at which it is run, an amount $\$p$ that each will pay in, and a deadline t_{report} by which $\mathcal{O}$ must deliver its report.

Finally, the two players agree with $\mathcal{O}$—still off-chain—on the terms of the oracle's reporting, namely $\mathsf{terms} = (S, t_{\mathsf{report}}, \mathcal{M})$. $\mathcal{O}$ sends signed terms $(\mathsf{terms}, \sigma_{\mathcal{O}}[\mathsf{terms}])$ to the players to signal agreement.[7]

Execution. In a subsequent *on-chain* execution phase performed by $\mathcal{M}$, one of the players sends the public agreed-upon values to $\mathcal{M}$, namely $(\mathcal{P}_0, \mathcal{P}_1)$ and $(\$p, t_{\mathsf{report}})$. The player also sends a commitment $\mathsf{commit}(\mathsf{terms})$; this is useful for regulatory assurance, as on-chain commitment of terms ensures that it can be presented in a correct and immutable form to a third party.

Each of the players then pays in $\$p$. Note that a player assents to the contract / binary option by paying in, only doing so if s/he agrees on the terms and has received signed terms from $\mathcal{O}$. $\mathcal{M}$ then waits for a report from $\mathcal{O}$.

$\mathcal{O}$ sends its report $s = \mathsf{switch}(x) \in \{0, 1\}$ to $\mathcal{M}$ once it is available.

Payment. In the final phase of SIMI, $\mathcal{M}$ makes payments to the players (or permits withdrawals, depending on the implementation). $\mathcal{O}$'s delivery of the report s assigns funds to the winning player, $\mathcal{P}_s$, who can immediately withdraw her / his funds.

[7]Alternatively, $\mathcal{O}$ might sign $(\mathsf{switch}, \mathcal{M})$ and $(t_{\mathsf{report}}, \mathcal{M})$ individually. In this way, a player can reveal t_{report} selectively to an auditor to prove non-performance by $\mathcal{O}$ without disclosing switch.

```
SIMI protocol (Parties: Alice, Bob, $\mathcal{O}$, and $\mathcal{M}$)

Setup (off chain):
1 :  Alice and Bob agree on $S$, $t_{\mathsf{report}}$, $\$p$, and (randomly ordered) pseudonyms $(\mathcal{P}_0, \mathcal{P}_1)$.
2 :  Alice (or Bob) sends $\mathsf{terms} = (S, t_{\mathsf{report}}, \mathcal{M})$ to $\mathcal{O}$.
3 :  $\mathcal{O}$ sends $(\mathsf{terms}, \sigma_{\mathcal{O}}[\mathsf{terms}])$ to Alice and Bob.

Execution (on chain, performed by $\mathcal{M}$):
1 :  $\mathcal{M}$ receives $\big((\mathcal{P}_0, \mathcal{P}_1), (\$p, t_{\mathsf{report}}), \mathsf{commit}(\mathsf{terms})\big)$ from Alice or Bob.
2 :  $\mathcal{M}$ receives $\$p$ from Alice and Bob.
3 :  $\mathcal{M}$ receives report $s = \mathsf{switch}(x)$ from $\mathcal{O}$.

Payment (on chain, perfomed by $\mathcal{M}$):
1 :  If report $s$ delivered and both players paid in $\$p$ to $\mathcal{M}$
2 :      $\mathcal{M}$ pays $\$2p$ to $\mathcal{P}_s$;
3 :  Else if $\textsc{time} > t_{\mathsf{report}}$
4 :      $\mathcal{M}$ refunds any paid-in funds.
```

Figure 3: Sketch of SIMI protocol for binary financial instrument.

If no report arrives by timeout t_{report}, then the players' payments to the Mixicle are refunded. Similarly, any payments are refunded if one player fails to pay in $\$p$.

We present a sketch of SIMI in Fig. 3. We assume for simplicity here and later that the identities of Alice, Bob, and $\mathcal{O}$ are hardwired into $\mathcal{M}$, although an implementation is possible of course in which they are set dynamically.

4.1.2 Security of SIMI

SIMI achieves mix-like payment-destination confidentiality: Thanks to the off-chain address randomization, neither external players nor $\mathcal{O}$ knows which of Alice and Bob received the funds from $\mathcal{M}$. SIMI doesn't achieve payment-amount confidentiality. An external observer can determine the payment amounts involved in $\mathcal{M}$.

SIMI also achieves query confidentiality: An external observer sees the oracle return nothing but a bit s.[8]

[8]Note that switch may have a bias—i.e., $\Pr[\mathsf{switch}(x) = 0] - \Pr[\mathsf{switch}(x) = 1] \neq 0$ over coins

What about the other three properties we're after: Auditability, collusion-resistance, and denial-of-service resistance?

Auditability results from $\mathcal{O}$'s commitment to terms. If $\mathcal{O}$ provides an incorrect report s, i.e., one that is inconsistent with switch, then either Alice or Bob can reveal terms, signed by $\mathcal{O}$, to an auditor, which can then confirm the discrepancy.

SIMI clearly provides collusion-resistance. If $\mathcal{O}$ reports a correct result before the deadline t_{report}, a valid payout will occur.

Finally, once the Mixicle is set up in SIMI, the only possible denial-of-service is failure by $\mathcal{O}$ to deliver a report. The timeout t_{report} ensures that players' payouts are refunded in this case.

4.2 SIMI-PS: Simple Mixicle with Payout Slates

While useful, SIMI reveals a key piece of information to external players: Pay-in and payout amounts. These amounts can leak important information. It's not just their magnitude that is important, but also their relative sizes. For example, if Alice is paying \$2 into binary option, while Bob is paying only \$1, that means Alice is taking the riskier position (2:1 odds). Similarly, if both players pay in \$1, and there is only one payout of \$2, it's clear that one player received all of the money from the Mixicle.

Happily, by generalizing SIMI, we can enforce good confidentiality for payment amounts—while still avoiding any heavyweight crypto. The idea is to generalize the pseudonymous payments made by the Mixicle.

We use the term *payout slate* to denote a sequence of pseudonyms and associated payment amounts. Let the pair $(\mathcal{P}, \$p)$ denote a payout of $\$p$ to pseudonym $\mathcal{P}$.

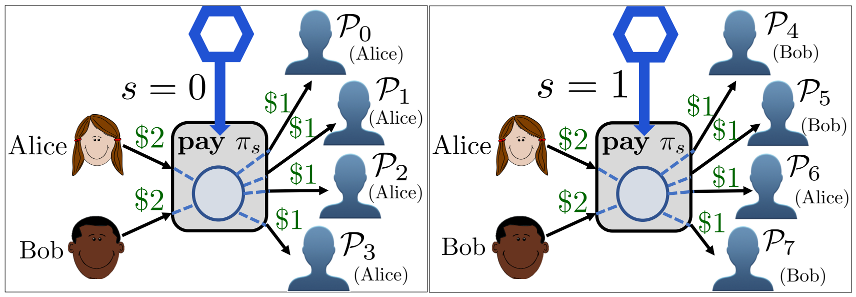
We represent a Mixicle payout slate as a set $\pi = \{(\mathcal{P}_i, \$p_i)\}_{i=0}^{n-1}$ of payouts over a set of some n accounts. (For simplicity, we assume n is fixed across payout slates.) We define $\pi[k] = (\mathcal{P}_i, \$p_k)$. As a Mixicle involves multiple slates, we index slates by subscript, i.e., π_i is the i^{th} payout slate, and $\pi_i[k]$ is the k^{th} payment in the i^{th} payout slate. We assume that all payout slates have equivalent aggregate value, and let thus $\$\pi$ denote the value of any payout slate.

By using a payout slate and also making *dummy payments* into the Mixicle to pad input amounts, players in a Mixicle can provide (partial) privacy for input amounts.

This idea is illustrated in the example in Fig. 4, in which payout slates with $n = 4$ are associated with the two possible outcomes $s = \mathsf{switch}(x) \in \{0, 1\}$. Here, $\pi_0 = \{(\mathcal{P}_i, \$1)\}_{i=0}^{3}$ and $\pi_1 = \{(\mathcal{P}_i, \$1)\}_{i=4}^{7}$.

In Fig. 4, an external player or $\mathcal{O}$ sees a financial instrument in which Alice and Bob pay in \$2 each, and the payout is in \$1 granularity—but nothing more. If $s = \mathsf{switch}(x) = 0$, the instrument here pays Alice \$4; in this case, Alice makes a

for X—and may thus leak data if reused. An easy fix can unbias switch. Alice and Bob pick a fresh random bit s' for each invocation of Mixicle contract $\mathcal{M}$ and replace switch with $\mathsf{switch}'(x) = \mathsf{switch}(x) \oplus s'$. This technique can obviously be extended to multi-valued functions switch. No change to $\mathcal{M}$ is required.



Payout slate π_0 Payout slate π_1

Figure 4: A binary option with payout slates π_0 and π_1. Payout slate π_0 pays \$4 to Alice, while π_1 pays \$1 to Alice and \$3 to Bob. Making payouts at dollar granularity, i.e., making four payouts of \$1 each, makes confidential with respect to external players and $\mathcal{O}$ the amounts received by Alice and Bob.

profit of \$2, while Bob loses \$2. If $s = 1$, then Bob receives \$3; thus Bob makes a profit of \$1, while Alice loses \$1. To put it another way, the profit / loss profile of the instrument is (Alice, Bob) $\leftarrow \big((+\$2, -\$2), (-\$1, +\$1)\big)$. Bob is taking a riskier position than Alice.

From the point of view of an external player or $\mathcal{O}$, though, other non-trivial, dollar-granularity profit/loss profiles are equally viable. It is equally plausible that Alice is taking the riskier position (Alice, Bob) $\leftarrow \big((-\$2, +\$2), (+\$1, -\$1)\big)$, or the two players could be assuming equal risk with symmetric positions, e.g., (Alice, Bob) $\leftarrow \big((-\$2, +\$2), (+\$2, -\$2)\big)$.

In general, payout slates are especially easy to work in the two-player Mixicle setting because of two special properties: *internal transparency* and *theft-proofness*.

- *Internal transparency:* In a two-player Mixicle, Alice knows that any input payments not output to her must have gone to Bob. So Bob has no secrets from Alice, and vice versa. We capture this idea by saying that Mixicle is *internally transparent*.

  If there were three players, Alice, Bob, and Carol, then internal confidentiality, instead of internal transparency, would be desirable. For example, Alice and Bob might want to keep private from Carol which of the two of them received funds from the Mixicle. (We are currently exploring such generalizations.)

  Because of its internal transparency, there is no loss in confidentiality if both Alice and Bob learn the payout slate in a Mixicle.

- *Theft-proofness:* Alice and Bob cannot specify a Mixicle payout slate that defrauds the other player. This is simply because they both agree to the slate. Additionally, they cannot defraud any other player, since they are the only players providing funds to $\mathcal{M}$.

Thanks to these two properties, to execute a Mixicle fairly and securely, Alice and Bob just have to agree between themselves on switch and Mixicle's payout slates. Thus a binary financial instrument can be defined by a switch with specification S, and a pair of payout slates $\vec{\pi} = (\pi_0, \pi_1)$, with $\$\pi_0 = \$\pi_1$. On receiving oracle input $s = \mathsf{switch}(x)$, the Mixicle contract $\mathcal{M}$ allows withdrawals against payout slate π_s.

A sketch of $\mathsf{SIMI\text{-}PS}$ is shown in Fig. 5. The protocol includes another small enhancement not in SIMI, an explicit deadline t_{setup} on players' commitments to funds distinct from the timeout t_{report}. This is because in practice, it may be desirable for t_{setup} to expire relatively quickly—with committed funds being refunded well before time t_{report}, should one player be faulty.

Payout Slates with Perfect Payout Privacy. As we have seen, how a payout slate is constructed critically determines how much information about payout amounts a Mixicle leaks to external players and $\mathcal{O}$. Using a large n in payout slates can help provide privacy for payout amounts, but results in high gas costs in Ethereum.

Here we show how to construct an efficient payout slate π, i.e., with small n, that has what we call *perfect payout privacy*, i.e., it reveals no information about payouts beyond the aggregate amount $\$\pi$. We show how to do this for \$1-granularity contracts, but our techniques can be generalized to any granularity. Viewed another way, our construction achieves confidentiality as strong as the "one dollar (\$1), one payment" approach exemplified in Fig. 4. The construction we now show achieves $n \leq \lceil \log_2 \$\pi \rceil$.

Suppose for simplicity that for payout slate π, we have $\$\pi = \$2^i - 1$. Writing the binary representation $\$\pi = \overbrace{11\ldots1}^{\ell}$ shows that $\$\pi = \sum_{i=0}^{\ell-1} \$\pi[i]$ for $\$\pi[i] = \$2^i$. In other words, $\$\pi$ can be composed of a set $D = \{\$\pi[i]\}_{i=0}^{\ell-1}$ of ℓ payments, each representing a '1' bit in the binary representation of $\$\pi$.

It follows that a payout slate π for any pair of payouts $(\$p_0, \$p_1)$ respectively to $\mathcal{P}_0$ and $\mathcal{P}_1$ such that $\$\pi = \$p_0 + \$p_1$ can be constructed using this same set D. Since $\$p_0 \leq \$\pi$, it can be written in binary as a string $\$p_0 = b_{\ell-1}\, b_{\ell-2}, \ldots, b_0$ (possibly with leading zeros). Thus, letting $D_0 = \{\$\pi[i]\}_{i \,|\, b_i = 1} \subseteq D$, we observe that $\$p_0 = \sum_{\$d \,\in\, D_0} \$d$ and $\$p_1 = \sum_{\$d \,\in\, D - D_0} \$d$.

As a result, we can always construct a payout slate π using the set D, and thereby ensure that π *leaks no information about payouts to the two players* (apart from $\$\pi$).

It is possible in general, of course, to round players' joint input amounts to a contract up to $\$2^\ell - 1$ for some ℓ and use the construction we've just described. We show in Appendix B.2, however, how to avoid the extra required monetary flow of such rounding by building an efficient payout slate with perfect payout privacy for any desired input amounts. It's also possible to extend our general approach to more than two players.

SIMI-PS protocol (Parties: Alice, Bob, $\mathcal{O}$, and $\mathcal{M}$)

```
Setup (off chain):
1 :  Alice and Bob agree on $S$, $t_{\mathsf{setup}}$, $t_{\mathsf{report}}$, and payout slate set $\vec{\pi} = (\pi_0, \pi_1)$.
2 :  Alice (or Bob) sends $\mathsf{terms} = (S, t_{\mathsf{report}}, \mathcal{M})$ to $\mathcal{O}$.
3 :  $\mathcal{O}$ sends $(\mathsf{terms}, \sigma_{\mathcal{O}}[\mathsf{terms}])$ to Alice and Bob.

Execution (on chain, performed by $\mathcal{M}$):
1 :  $\mathcal{M}$ receives $\big(t_{\mathsf{setup}}, t_{\mathsf{report}}, \vec{\pi}, \mathsf{commit}(\mathsf{terms})\big)$ from Alice (or Bob).
2 :  $\mathcal{M}$ sets $\mathsf{payinFailure} \leftarrow \mathtt{false}$.
3 :  $\mathcal{M}$ receives $\$\pi/2$ from Alice and Bob.
4 :  If $\textsc{time} > t_{\mathsf{setup}}$ and one player has not paid in $\$\pi/2$,
5 :      $\mathcal{M}$ sets $\mathsf{payinFailure} \leftarrow \mathtt{true}$; break
6 :  $\mathcal{M}$ receives $s = \mathsf{switch}(x)$ from $\mathcal{O}$.

Payment (on chain, performed by $\mathcal{M}$):
1 :  If $\mathsf{payinFailure} = \mathtt{true}$
2 :      $\mathcal{M}$ refunds paid-in funds.
3 :  If report $s$ delivered
4 :      $\mathcal{M}$ pays against payout slate $\pi_s$;
5 :  Else if $\textsc{time} > t_{\mathsf{report}}$
6 :      $\mathcal{M}$ refunds paid-in funds.
```

Figure 5: Protocol sketch of SIMI-PS for binary financial instrument. Here, TIME denotes the current time.

5 Main Construction: SIMI-PPS

We are now ready to introduce our main construction, *Simple Mixicle with Private Payout Slates*, or SIMI-PPS.

SIMI-PPS permits players to engage in a *sequence* of financial-instrument executions. We refer to each such execution in SIMI-PPS as a *round*. In a basic version of SIMI-PPS, players engage in multiple rounds of interaction, i.e., execution of multiple financial instruments. They withdraw funds, however, only in the last, or *terminal* round. We show how SIMI-PPS can *enable private payouts in all non-terminal rounds*, i.e., make intermediate state private from both external players and the oracle.

SIMI-PPS also supports switches with more than two outputs in SIMI-PPS, i.e., cases where the number of possible oracle reports, which we call the *switch cardinality*, is non-binary. In other words, SIMI-PPS allows for $\mathsf{switch}(x) \in \mathbb{Z}_q$ for some $q > 1$. Since q may vary across Mixicle (and even rounds), SIMI-PPS therefore also *keeps the switch cardinality q private* from external players. (We also show how to achieve weak privacy with respect to the oracle.)

In this section, we first briefly explain our two key ideas in SIMI-PPS: How to make non-terminal payout slates private (Section 5.1) and how to make switch cardinality private (Section 5.2). We then give a design overview over SIMI-PPS (Section 5.3) before giving a detailed account of our construction (Section 5.4). Finally, we discuss regulatory compliance and auditing (Section 5.5) as well as some further variants (Section 5.6), and present our prototype implementation including benchmarks (Section 5.7).

5.1 Privacy for non-terminal payout slates

In SIMI-PPS, players engage in multiple rounds, each corresponding to the execution of a financial instrument. We use index j to denote a round and denote by $v^{(j)}$ a given value v in round j, for $j \in \{1, 2, \ldots\}$.

To make payout slates in non-terminal rounds in SIMI-PPS private, players do not send these slates explicitly to $\mathcal{M}$. Instead, they commit *off chain* to what we call a *payout slate set*, and denote by $\vec{\pi}^{(j)}$. This is the set of possible payout slates in a given round j of the Mixicle.

Specifically, players commit to $\vec{\pi}^{(j)}$ by signing each payout slate it contains individually. That is, defining $\vec{\pi}_i^{(j)} \triangleq \pi_i^{(j)}$ as the i^{th} possible slate in round j, players sign each of $\{\pi_i^{(j)}\}_{i=0}^{n-1}$ (along with their associated round numbers).

The payout slate that emerges as valid from a given round j, which we denote by $\pi^{(j)}$, is determined by the oracle report $s^{(j)}$ for the round. That is, $\pi^{(j)} = \pi_{s^{(j)}}^{(j)}$.

Only in the terminal round do players reveal to $\mathcal{M}$ the final (signed) payout slate. This payout slate, $\pi_{s^{(j)}}^{(j)}$, we denote also by π_{term}. Players make withdrawals against π_{term}.

Players can make additional payments into the Mixicle between rounds. (We discuss partial withdrawals later.)

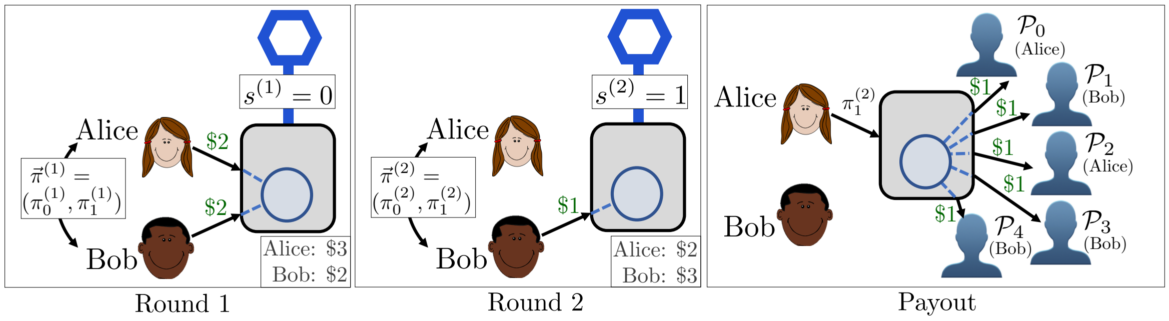


Figure 6: Illustration of Example 1. Mixicle with private payout slates. The table on the bottom right of the panels for round 1 & 2 shows the private balances of the players at the end of the round.

Example 1. We convey the basic idea of payout slate privacy through a simple example in Fig. 6 involving binary financial instruments.

Here, Alice and Bob begin by each paying in \$2. Off chain, they execute financial instrument $F^{(1)}$, with payout slate set $\vec{\pi}^{(1)}$, which consists of two slates, as the instrument is a binary one. Bob loses \$1 in this round, so Alice and Bob end up with respective balances \$3 and \$1. At the end of the execution in Round 1, *no information about balances or payout slates is revealed, i.e., sent to* $\mathcal{M}$.

The players then begin a new round. They agree on a new financial instrument $F^{(2)}$ with payout slate set $\vec{\pi}^{(2)}$. This instrument requires that Bob pay in \$1. After this instrument is executed in Round 2, Alice loses \$1 to Bob, leaving the players with respective balances \$2 and \$3.

After $F^{(2)}$ is executed, Alice chooses to terminate the Mixicle. Alice reveals the terminal payout slate $\pi_{\mathsf{term}} = \pi_1^{(2)}$, the currently valid payout slate. The contract $\mathcal{M}$ checks that this slate matches the most recent oracle report $s^{(2)}$, and then permits the players to withdraw their funds.

As we observe in Example 1, external players and $\mathcal{O}$ see *only the payouts in* π_{term}. They observe neither unexercised payout slates nor the intermediate balances of the two internal players. The result is privacy for intermediate funds much like that accomplished in, e.g., Zether [19] or in state channels, but without the use of expensive cryptography. Obviously the idea can be extended to an arbitrary number of rounds.

If a fault occurs at any time in SIMI-PPS, one player can initiate withdrawals against the current payout slate. For example, if Bob or $\mathcal{O}$ fails to follow through on the execution of a financial instrument in round j or the setup in round $j+1$, Alice can terminate the Mixicle and make withdrawals against the currently valid payout slate $\pi^{(j)} = \pi_{\mathsf{term}}$.

5.2 Private switch cardinality

Given the use of signed payout slates, it is easy to make switch cardinality q private with respect to external players.

The basic idea here is quite simple. The report $s^{(j)} \in \mathbb{Z}_q$ is an index into $\vec{\pi}^{(j)}$, and thus leaks information about q. We therefore now modify the output of $\mathcal{O}$ by replacing $s^{(j)}$ with a random value (bitstring) that we call a *tag*. Tags, unlike indexed reports, do not leak information about q.

To give further detail, the players assign to each possible value $i \in \mathbb{Z}_q$ of $s^{(j)}$ a corresponding tag $\tau_i^{(j)}$. This tag $\tau_i^{(j)}$ can, for example, be a random[9] 128-bit bitstring, i.e., $\tau_i^{(j)} \xleftarrow{\$} \{0,1\}^{128}$. Tags are used by $\mathcal{O}$ in place of indices when sending a report. That is, instead of sending $s^{(j)} = \mathsf{switch}^{(j)}(x^{(j)})$, $\mathcal{O}$ sends the tag $\tau_{s^{(j)}}^{(j)}$.

We give further details in our specification of SIMI-PPS on how tags are used.

Making switch cardinality private with respect to $\mathcal{O}$. The players may further make the switch cardinality and financial instrument structure private with respect to $\mathcal{O}$ by introducing *dummy outcomes*. Consider a binary option involving Tesla's stock price sp, specifically whether $\mathsf{sp} \geq \$1000$. The players can ask $\mathcal{O}$ to report the stock price rounded down to the nearest \$100. They can create and send to $\mathcal{O}$ signed tags associated with each such outcome for sp (say for {\$0, \$100, ..., \$2000}), but generate and associate with these outcomes only two distinct payout slates for the binary option, π_0 for $\mathsf{sp} < \$1000$ and π_1 for $\mathsf{sp} \geq \$1000$. In this way, players can reveal to $\mathcal{O}$ only an upper bound $q' \geq q$ on the true switch cardinality q (e.g., $q' = 21$ and $q = 2$ in our short example here).

5.3 Design overview

SIMI-PPS is designed to meet the confidentiality and security goals outlined in Section 3. It thus brings together all of the confidentiality techniques we have described, including privacy for non-terminal payout slates and switch cardinality (as well as payout slates with perfect payout privacy). SIMI-PPS reveals to external players only: (1) The amounts *paid in* by players, which is equal to the *total amount* $\$\pi_{\mathsf{term}}$ paid out to players; (2) The number of rounds of execution; (3) The timing of oracle reports and withdrawals. It reveals to $\mathcal{O}$ only these three pieces of information plus $\mathsf{switch}^{(j)}$ for every round j.

SIMI-PPS also achieves robustness to oracle faults as described in Section 3 by allowing players to withdraw funds when $\mathcal{O}$ responses time out and by permitting players to prove malfeasance by $\mathcal{O}$ to third parties such as auditors.

An important operational design goal in SIMI-PPS is to move logic off-chain where possible, in the same spirit as state channels [30, 32, 66, 70]. To ensure oracle au-

[9]As a standard optimization, tags may be pseudorandom, i.e., $\tau_s^{(j)} = f_\kappa(s, j)$ for a pseudorandom function f, where κ is a seed agreed-upon by internal players.

ditability, however, and to preserve simplicity, SIMI-PPS dispenses with the on-chain dispute handling mechanisms and off-chain monitoring functionality [69, 61] required to handle faults and cheating in state-channel schemes. In SIMI-PPS, oracle reports, which trigger state updates, are sent to the contract $\mathcal{M}$. Hence, unlike a contract for a state channel, $\mathcal{M}$ can autonomously identify and arbitrate *current* state within a Mixicle. (For comparison, we briefly discuss a state-channel variant of Mixicle in Section 5.6.)

Furthermore, we minimize the number of contract calls to save per-call gas overheads.

5.4 SIMI-PPS: Construction details

We are now ready to present our construction of SIMI-PPS. We present a two-player variant, as above. To simplify notation, we leave implicit the inclusion of the round index and the address of $\mathcal{M}$ in signed messages and commitments. Fig. 7 shows a sketch of the protocol and Fig. 8 shows the protocol state machine implemented by the Mixicle smart contract.

In each round j of a SIMI-PPS Mixicle, players select and execute a fresh financial instrument $F^{(j)}$. This instrument consists of

- a switch specification, which we denote by $S^{(j)} = (\mathsf{switch}^{(j)}, X^{(j)})$.
- a setup deadline $t^{(j)}_{\mathsf{setup}}$. This deadline prevents issues around (un)fair exchange by forcing the setup to be completed by a certain time.
- an oracle report delivery deadline $t^{(j)}_{\mathsf{report}}$.
- a payment slate set $\vec{\pi}^{(j)}$. (As a reminder, each individual payout, i.e., $\pi^{(i)}_j[k]$, includes a payee pseudonym / address.)
- a ordered set of random tags $T^{(j)} = \{\tau^{(j)}_i\}^q_{i=1}$ associated with the q possible outcomes for the instrument as described above in Section 5.2. Recall that each possible outcome corresponds to a switch value $s^{(j)} = \mathsf{switch}^{(j)}(x^{(j)}) \in \mathbb{Z}_q$, and this value selects the payout slate $\pi^{(j)} = \pi^{(j)}_{s^{(j)}}$ that becomes valid for the round, i.e., is the round's end state. (The idea of oracle reports selecting payout slates remains the same as in our warm-up construction SIMI-PS, as shown in Fig. 4.)

Thus $F^{(j)} = (S^{(j)}, t^{(j)}_{\mathsf{setup}}, t^{(j)}_{\mathsf{report}}, \vec{\pi}^{(j)}, T^{(j)})$. A round proceeds in two phases, *round setup* and *round execution.* A final phase of execution, which we call *termination and payment*, can be triggered in any given round, as we soon explain.

Round setup. To initiate a round, the players first agree upon $F^{(j)}$ off chain and generate a payout slate set enhanced with the tags from T, denoted by $\vec{\pi}_{+\tau}^{(j)} = \{(\pi_i^{(j)}, \tau_i^{(j)})\}_{i=1}^{q}$. They additionally both sign each element of $T^{(j)}$ and $\vec{\pi}_{+\tau}^{(j)}$; in a slight abuse of notation, we let $\sigma[T^{(j)}]$ and $\sigma[\vec{\pi}_{+\tau}^{(j)}]$ respectively denote these sets of signatures.

The players then send a proposal $\mathsf{proposal}^{(j)} = (S^{(j)}, t_{\mathsf{setup}}^{(j)}, t_{\mathsf{report}}^{(j)}, \$\pi^{(j)}, T^{(j)}, \sigma[T^{(j)}])$ along with signatures $\sigma[\mathsf{proposal}^{(j)}]$ to $\mathcal{O}$. If $\mathcal{O}$ approves of $\mathsf{proposal}^{(j)}$, the players and $\mathcal{O}$ agree on terms $\mathsf{terms}^{(j)} = (S^{(j)}, T^{(j)})$ and a corresponding commitment $\mathsf{commit}(\mathsf{terms}^{(j)})$ to which all parties know the opening, as well as smart contract parameters $\mathsf{params}^{(j)} = (t_{\mathsf{setup}}^{(j)}, t_{\mathsf{report}}^{(j)}, \$\pi^{(j)}, \mathsf{commit}(\mathsf{terms}^{(j)}))$ with a signature by all parties $\sigma[\mathsf{params}^{(j)}]$. Having agreed on all details of the round, one of player sends $\mathsf{params}^{(j)}$ and $\sigma[\mathsf{params}^{(j)}]$ to $\mathcal{M}$ prior to the expiry of $t_{\mathsf{setup}}^{(j)}$. If $\mathcal{M}$ isn't fully funded yet (i.e. the balance of $\mathcal{M}$ is less than $\$\pi^{(j)}$), both players can pay in the required funds until $t_{\mathsf{setup}}^{(j)}$ expires. Round setup is complete once sufficient funds are present.

Round execution. The completion of a round is triggered by delivery of a report from $\mathcal{O}$ to $\mathcal{M}$.

This report takes the form of a tag $\tau^{(j)} = \tau_{s^{(j)}}^{(j)}$ representing the output of the switch $s^{(j)} = \mathsf{switch}^{(j)}(x^{(j)})$ in $F^{(j)}$. Recall use of a tag in lieu of switch output $s^{(j)}$ makes the switch cardinality q private. $\mathcal{O}$ also sends the players' signature $\sigma[\tau^{(j)}]$ so that $\mathcal{M}$ can confirm the validity of the tag.[10] (This prevents a malicious oracle from mounting a denial-of-service attack using an invalid tag.)

The players hold (off chain) a pair $(\pi^{(j)}, \tau^{(j)})$ with signature $\sigma[(\pi^{(j)}, \tau^{(j)})]$ created during the round setup. This pair creates a binding between the tag $\tau^{(j)}$ and $\pi^{(j)}$, validating $\pi^{(j)}$ as the current state of the Mixicle.

In a non-terminal round, $\pi^{(j)}$ is not revealed to $\mathcal{M}$, and remains confidential.

Termination and payment. Round j is terminal, if (1) Either player calls the `payout` function of $\mathcal{M}$ while $\mathcal{M}$ isn't currently awaiting a report from $\mathcal{O}$; (2) A failure occurs during the setup of round $j+1$ because the balance of $\mathcal{M}$ doesn't reach $\$\pi^{(j+1)}$ before $t_{\mathsf{setup}}^{(j+1)}$ expires; or (3) $\mathcal{O}$ fails to deliver a report before $t_{\mathsf{report}}^{(j+1)}$ expires.

More complex rules, e.g., with $\mathcal{M}$ enforcing termination by some predetermined round $j = r_{\mathsf{term}}$ are also possible.

Upon invocation of this rule, one player must send $(\pi^{(j)}, \tau^{(j)})$ along with signatures by both players generated during setup $\sigma[(\pi^{(j)}, \tau^{(j)})]$ to $\mathcal{M}$. $\mathcal{M}$ verifies that $\tau^{(j)}$

[10] As an optimization, we can avoid submitting this signature in the optimistic case as follows: Tags can be ordered and indexed by increasing value, creating an authenticated dictionary. To prove to $\mathcal{M}$ that it has received an invalid tag $\tilde{\tau}$, then, a player may send $\mathcal{M}$ a pair of successive tags, signed with their indices, $\big((\tau_e, e), (\tau_{e+1}, e+1)\big)$, such that $\tau_e < \tilde{\tau} < \tau_{e+1}$.

As players authenticate tags by signing them, proof of malfeasance by $\mathcal{O}$ to an auditor also requires reference to $\mathsf{terms}^{(j)}$, which as noted above, includes tags in SIMI-PPS.

matches the lates outcome tag reported by $\mathcal{O}$. If so, $\pi^{(j)} = \pi_{\mathsf{term}}$ is confirmed as the terminal slate. $\mathcal{M}$ then pays players against payout slate π_{term}.

Failure modes. The use of digital signatures prevents either player, even in collusion with $\mathcal{O}$, from instantiating an invalid state $\pi^{(j)}$ in $\mathcal{M}$ provided that $\mathcal{O}$ provides a correct report. If $\mathcal{O}$ sends an incorrect report, then its misbehavior can be proven during an audit, as explained in Section 4.1.2.

Players and $\mathcal{O}$ *can*, however, delay or fail to send messages, resulting in unfair execution or a denial of service. This can happen at any of the following ways. (For simplicity, we assume that Bob is the offending player.)

- *Setup delay:* Alice proceeds promptly through the setup (e.g. sending signed messages and funding $\mathcal{M}$), but Bob stalls, delaying the next step he needs to perform. Bob thus gains an advantage. He can, for instance, wait to see whether the financial instrument is developing favorably; if not, he permanently withholds his signatures.

- *Execution delay:* $\mathcal{O}$ delays delivery of its report or never delivers it. $\mathcal{O}$ could thus, for instance, collude with Bob to defraud Alice if it looks like the current round will favor her.

Our design for SIMI-PPS here addresses setup delays through the setup deadline t_{setup}. If setup isn't completed by the deadline, both players can withdraw any funds they added to $\mathcal{M}$ during setup and payout for the previous round (if one exists) is triggered. t_{setup} can be flexibly chosen depending on the needs of the players, e.g., based on the urgency of the instrument being set up and the instrument's time horizon. More involved mechanism involving, for example, forfeiture of staked funds could also be implemented.

Execution delays are addressed using the mechanism described in Section 4.1.2 for failure by $\mathcal{O}$ to deliver a report.

5.5 Regulatory compliance / auditing

SIMI-PPS, like our warm-up constructions SIMI and SIMI-PS, naturally commits data on chain in a way that facilitates trustworthy reporting of the behavior of a Mixicle to a third party $\mathcal{T}$, be it an auditor or regulator.

Commitment to terms enables presentation of a correct, immutable record of the $\mathcal{O}$ input to $\mathcal{T}$, specifically switch. Either player—or $\mathcal{O}$—can decommit terms for $\mathcal{T}$. By (optionally) including including $\vec{\pi}$ in terms the presentation of the instrument's payout options is similarly possible. Players can prove ownership of the pseudonyms $\{\mathcal{P}_i\}_{i=0}^{n-1}$ associated with each payout slate using their private keys (e.g., signing challenges from $\mathcal{T}$). Together, switch, $\vec{\pi}$, and the identities of pseudonym owners define a Mixicle's

```
SIMI-PPS protocol

Setup (Off chain, Round $j$):
1 : Alice and Bob agree on financial instrument $F^{(j)} = (S^{(j)}, t^{(j)}_{\mathsf{setup}}, t^{(j)}_{\mathsf{report}}, \vec{\pi}^{(j)}, T^{(j)})$.
2 : Alice and Bob jointly generate $\vec{\pi}^{(j)}_{+\tau}$, $\sigma[\vec{\pi}^{(j)}_{+\tau}]$, and $\sigma[T^{(j)}]$.
3 : Alice and Bob prepare $\mathsf{proposal}^{(j)} = (S^{(j)}, t^{(j)}_{\mathsf{setup}}, t^{(j)}_{\mathsf{report}}, \$\pi^{(j)}, T^{(j)}, \sigma[T^{(j)}])$.
4 : Alice and Bob send $\mathsf{proposal}^{(j)}$ with signature $\sigma[\mathsf{proposal}^{(j)}]$ to $\mathcal{O}$.
5 : Alice, Bob, and $\mathcal{O}$ agree on $\mathsf{params}^{(j)} := (t^{(j)}_{\mathsf{setup}}, t^{(j)}_{\mathsf{report}}, \$\pi^{(j)}, \mathsf{commit}(\mathsf{terms}^{(j)}))$.
6 : Alice, Bob, and $\mathcal{O}$ jointly generate signatures $\sigma[\mathsf{params}^{(j)}]$.

Execution (On chain, Round $j$):
1 : Alice sends $\mathsf{params}^{(j)}$ and $\sigma[\mathsf{params}^{(j)}]$ to $\mathcal{M}$.
2 : Alice and/or Bob (optionally) pays into $\mathcal{M}$ (in round 1 or if $\$\pi^{(j-1)} < \$\pi^{(j)}$).
3 : $\mathcal{O}$ finds $s^{(j)} = \mathsf{switch}^{(j)}(x^{(j)})$ and sends $\tau^{(j)}_{s^{(j)}}$ with $\sigma[\tau^{(j)}_{s^{(j)}}]$ to $\mathcal{M}$.

Termination (Round $j$):
1 : Alice sends a decommitment of $\pi_{\mathsf{term}} = \vec{\pi}^{(j)}_{s^{(j)}}$ with $\sigma[\vec{\pi}^{(j)}_{s^{(j)}}, \tau^{(j)}_{s^{(j)}}]$ to $\mathcal{M}$.
2 : $\mathcal{M}$ makes (pseudonymous) payouts against payout slate $\pi_{\mathsf{term}}$.
```

Figure 7: Sketch of SIMI-PPS protocol. We assume for simplicity that Alice initiates communication with $\mathcal{M}$ and $\mathcal{O}$. In general, either player could, and pseudonymous players can decommit π_{term}.

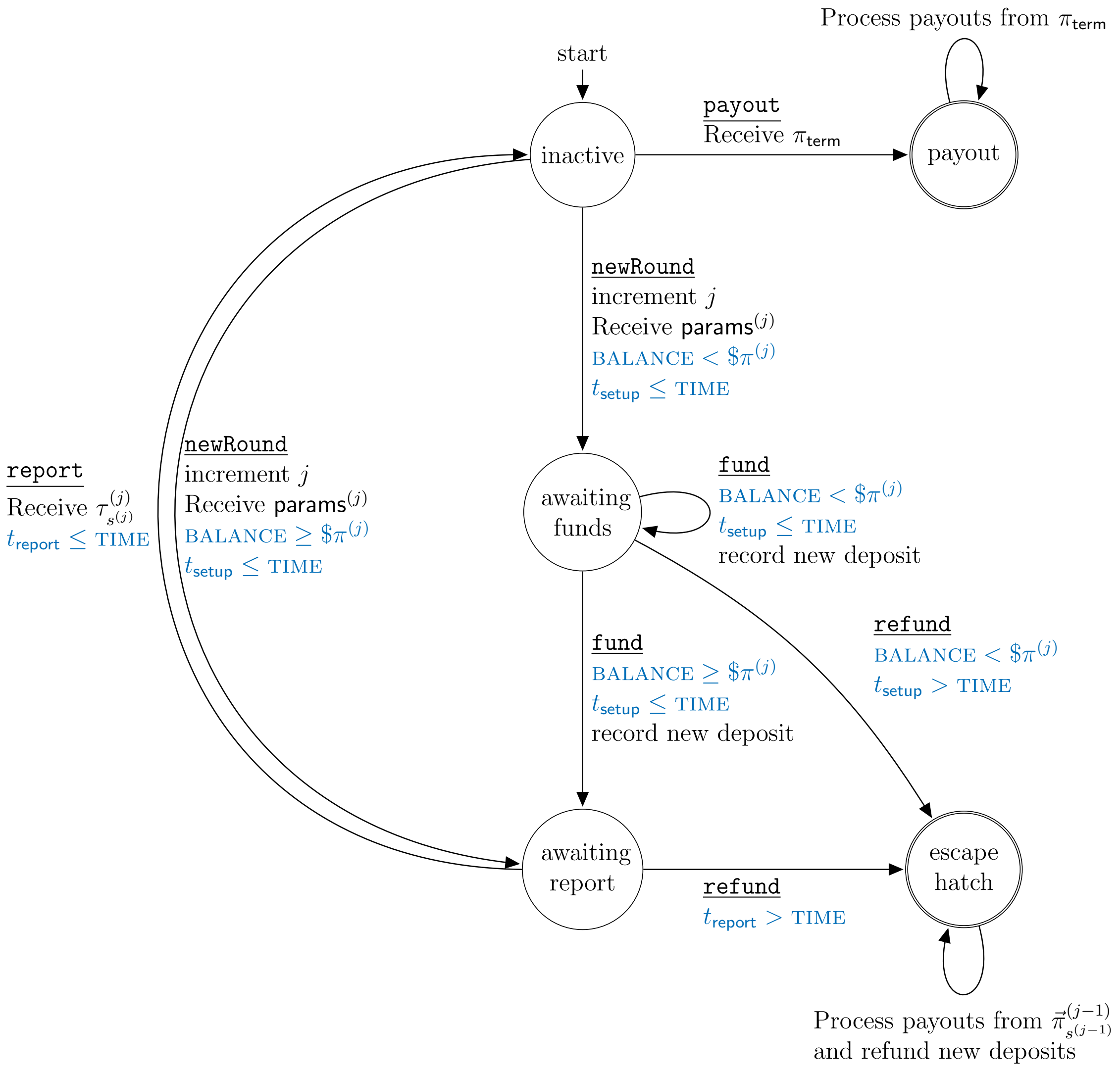

Process payouts from π_{term}
start
inactive
payout
Receive π_{term}
payout
newRound
increment j
Receive $\mathsf{params}^{(j)}$
$\textsc{balance} < \$\pi^{(j)}$
$t_{\mathsf{setup}} \leq \textsc{time}$
report
Receive $\tau_{s^{(j)}}^{(j)}$
$t_{\mathsf{report}} \leq \textsc{time}$
newRound
increment j
Receive $\mathsf{params}^{(j)}$
$\textsc{balance} \geq \$\pi^{(j)}$
$t_{\mathsf{setup}} \leq \textsc{time}$
awaiting funds
fund
$\textsc{balance} < \$\pi^{(j)}$
$t_{\mathsf{setup}} \leq \textsc{time}$
record new deposit
refund
$\textsc{balance} < \$\pi^{(j)}$
$t_{\mathsf{setup}} > \textsc{time}$
fund
$\textsc{balance} \geq \$\pi^{(j)}$
$t_{\mathsf{setup}} \leq \textsc{time}$
record new deposit
awaiting report
refund
$t_{\mathsf{report}} > \textsc{time}$
escape hatch
Process payouts from $\vec{\pi}_{s^{(j-1)}}^{(j-1)}$
and refund new deposits

Figure 8: Simplified state machine implemented by SIMI-PPS Mixicle contract. Function calls act as state transitions. We omit signature and caller identity checks from the figure, but highlight other conditions that have to be satisfied in order for a transition to be taken. TIME and BALANCE represent the current time and contract's balance respectively.

underlying DeFi. Such information can also be revealed *selectively*: A player can decommit only what she wants to report to $\mathcal{T}$.

Alternatively, if $\mathcal{T}$ is concerned about players being unavailable, terms can be placed on chain *encrypted* under the public key $\mathsf{pk}_{\mathcal{T}}$ of the auditor / regulator, as can a joint record of ownership of pseudonyms. This way, even if players go offline, the Mixicle remains completely transparent to $\mathcal{T}$.

One benefit of such encryption is that it can provide *conditional anonymity revocation* for law enforcement, a notion first proposed decades ago in connection with pre-Bitcoin anonymous cash [18, 79]. The idea is that a law enforcement entity $\mathcal{T}$ can request, e.g., a warrant granting permission to decrypt the ciphertext. Additionally, $\mathcal{T}$ can be *distributed*. Specifically, the private key $\mathsf{sk}_{\mathcal{T}}$ can be shared out among a set of n committee members constituting $\mathcal{T}$ such that any k can perform *threshold* decryption, as proposed in [44], to offer flexible but abuse-resistant law-enforcement access. For example, $\mathsf{sk}_{\mathcal{T}}$ might be distributed among three law enforcement agencies such that any two can decrypt Mixicle data, but no one such agency can in by itself.

Other approaches. Of course, there are many other possible approaches to regulatory oversight, e.g., $\mathcal{O}$ could require signoff from a regulator or enforce AML / KYC compliance for players before providing a report to a Mixicle.

Other enhancements are possible too. For example, $\mathcal{M}$ can incorporate various mechanisms for *control* by a regulator. For example, $\mathcal{M}$ might support modification or termination by regulators for cases of malfeasance by players or to manage systemic risks, or allow targeted human intervention, as suggested in, e.g., [26, 58].

We leave exploration of these possibilities to future work. We note, though, that in general, the existence of an audit trail on chain means *more robust audit / regulatory inspection for Mixicles than for existing, legacy financial systems.*

5.6 Variants on SIMI-PPS

Our basic protocol for SIMI-PPS can be enriched and modified in a number of ways, of which we mention a few.

- *Merkle-tree-based payout slate set commitments:* Players may commit to payout slates using Merkle trees, instead of signatures. In every round, they compute a Merkle tree of depth ℓ for $2^\ell \geq q$ whose leaves contain committed pairs $\{(\pi_0^{(j)}, \tau_0^{(j)}), (\pi_1^{(j)}, \tau_1^{(j)}), \ldots, (\pi_{q-1}^{(j)}, \tau_{q-1}^{(j)})\}$ in randomly assigned positions. Let $R^{(j)}$ denote the Merkle root of this tree. To decommit a payout slate, a player reveals its corresponding leaf and decommitment, along with a path to $R^{(j)}$ (in the usual manner). External players then learn only that $k \leq 2^\ell$, but *do not learn q itself.*[11]

[11]Our back-of-the-envelope calculation of relative gas costs in Ethereum is as follows. A signature is currently 3000 gas for the ECRECOVER call, plus about $68 \times (32 + 32 + 1) = 4420$ gas to pass

Function	Gas Cost
`newRound`	170599
`fund`	50011
`report/fullfill`	30058
`payout` (5 entries)	243085
`payout` (10 entries)	416206
`payout` (20 entries)	759954
`refund`	27666

Table 2: Gas costs of SIMI-PPS Mixicle prototype smart contract

- *Partial withdrawals:* SIMI-PPS can support partial withdrawals in non-terminal rounds. To preserve confidentiality on unwithdrawn funds, this variant would require fine-grained, authenticated disclosure of payouts, e.g., through signing at the granularity of individual payouts. Provided that players sign every payout in $\{\pi[k]\}_{k=1}^{n}$ for every payout slate π, individual payouts can be disclosed on chain without revealing the remainder of π.

- *State channels:* Another possible variant of SIMI-PPS involves implementation as a state channel. Such a variant can be implemented within a generalized state-channel framework, such as that of Miller et al. [66]. That work shows how to realize in Ethereum (in the Universal Composibility (UC) framework) an ideal functionality $\mathcal{F}_{\mathsf{State}}$ that executes a replicated state machine abstraction. This state machine can intercommunicate with parties and with an on-chain smart contract, and the framework is general enough to instantiate $\mathcal{O}$ as one of the parties. While we omit details, it is a straightforward matter to map any of our Mixicle protocols onto $\mathcal{F}_{\mathsf{State}}$.

  Apart from its implementation complexity (due in part to its generality), however, this state-channel approach has the drawback common to all state channels, a need for off-chain monitoring.

  Additionally, the approach is not compatible with existing Chainlink oracle infrastructure; it is at odds with the on-chain performance enforcement mechanisms. It also does not support regulatory compliance, an explicit goal of our constructions here. These last two limitations could be remedied in principle with additional features.

5.7 Implementation

We have implemented a SIMI-PPS Mixicle prototype smart contract for Ethereum in Solidity using Chainlink's SDK. Thanks to the intentional simplicity of our protocol, the contract consists of only 188 lines of code (excluding comments), making it easy to audit. (We would like to emphasize, however, that this prototype has not been audited and should not be used in production as is.) Furthermore, the gas cost of running the contract is moderate even though the contract has not been optimized. Table 2 gives a detailed breakdown of the gas costs of the prototype. For a realistic payout slate size of 10, a complete run of a single round of the protocol ($2 \times$ `fund`, `newRound`, `report`, `payout`) costs a total of 716885 gas. Note that in practice, we expect players to play several rounds, allowing us to amortize the costs of funding and payout over multiple rounds; e.g., running five rounds costs 1519513 gas, leading to a per-round cost of 303903 gas. The implemented contract slightly differs from the protocol described above in Section 5.4 due to some specifics of Chainlink's current architecture:

- Whenever a new round is being set up, the example contract makes a call to the Chainlink oracle contract to request delivery of data and pays the oracle in Chainlink's LINK tokens. Users are expected to have funded the example contract with an appropriate amount of tokens. (As we mention in Section 3.4, we have chosen for simplicity to omit payments to oracles from protocol descriptions in this paper.)

- Due to limitations of previous versions of the Ethereum Virtual Machine, Chainlink currently only allows oracles to deliver a 32 byte response payload. Therefore, in our prototype the oracle only delivers a tag value, but no player signatures, allowing a malicious oracle to report an invalid tag for which the players haven't agreed on a payout slate. We note that this attack is likely not too relevant in practice, since the oracle can already mount a more powerful attack by reporting a false outcome.

Code for our example implementation is currently available at `https://github.com/smartcontractkit/mixicles`. *We note again that this prototype code is meant to elucidate and test ideas in the paper. It has not been audited and shouldn't be deployed in production.*

the signature into the contract. To check a level in a Merkle proof is $68 \times 32 = 2176$ gas, the cost of passing the hash into the contract, plus $(30 + 6 + 6) = 42$ gas to compute the parent hash. So a signature is cheaper in terms of gas for $\ell > 7$.

6 Extensions and Future Directions

6.1 More complex financial instruments

We have thus far focused on binary options with fixed settlement times, a very basic financial instrument. The Mixicle framework, however, can support more complex instruments. Examples include:

- *Marking to market:* SIMI-PPS naturally supports financial instruments with marking to market, a term denoting daily settlement of changes in asset value. Each such daily settlement may be treated as a round. Players can pay into the instrument or withdraw on a round-by-round basis, as required. Of course, the Mixicle does not have a built-in mechanism to handle defaults, i.e., players not paying in as required. This, however, is a general problem for DeFi instruments, and can be handled, for example, by restricting instrument functionality in various ways to avoid counterparty risk (see, e.g., [35]). For example, funds potentially paid in for marketing to market might be capped by the instrument and then prepaid by a player and held in escrow by the Mixicle.
- *Stop orders:* A *stop order* is a trading order that executes when an asset reaches a target price. It is straightforward in the Mixicle framework to support financial instruments containing stop orders—or other event-triggered logic. $\mathcal{O}$ is simply required to send a report s when the stop is met (within delay t_{report}).

Perhaps the most common need in practice for Mixicles, though, will be a large space of possible outcomes.

Many outcomes. During setup of the protocols we discussed so far, players need to do $O(|X|)$ work, which can be burdensome for large outcome spaces associated with real-world instruments. Consider the example of a European call option on one share of GOOGL (currently trading at \$1179.21) with a strike price of \$1500 and a premium of \$700. To prudently account for large price swings, the players would want to cover every price at maturity up to, e.g., \$10,000. Fig. 9 shows the corresponding payout diagram. At a tick size of \$0.01 typical for NASDAQ, the players would have to generate and sign close to a million payout slates (1 for "price at maturity is below \$1500.00" and 850,000 to cover every price between \$1500.00 and \$10,000.00) — a cumbersome amount of computation.

Fortunately, the payout function of our example instrument as well as those of most commonly used real-world derivatives are piecewise linear functions of the price of the underlying asset. In this case, we can reduce the amount of off-chain computation by the players to $O(\sqrt{|X|})$ by pushing a little more work on-chain as we sketch in the following. For ease of exposition, we briefly return to the setting where the oracle reports switch values s (like in SIMI-PS) rather than randomly chosen tags

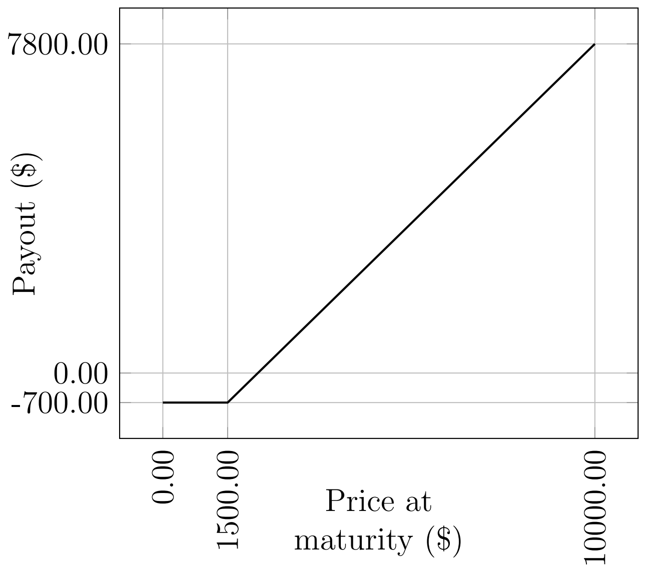


Figure 9: Payout diagram for a call option with a strike price of \$1500 and a premium of \$700.

(like in SIMI-PPS). We also make the simplifying assumption here that switch reveals x. (In any actual implementation, we would use tags as in SIMI-PPS for privacy.)

Instead of having switch return a single s, we instead have it return a tuple (s_1, s_2) that is a base-c representation of the price of the underlying asset x, where c is a constant base chosen by the players during setup (with the goal of minimizing the amount of computation the players have to perform). The first element is computed as $s_1 := \lfloor x/c \rfloor$, the second element is computed as $s_2 := x \bmod c$, and hence $x = s_1c + s_2$. For a linear piece[12] of our payout function $f(x) = ax + b$, we thus have $f(x) = as_1c + (as_2 + b)$, and we can have our players sign two *independent* payout slates for all possible values of s_1 and and s_2. $\mathcal{O}$ will report both s_1 and s_2 to $\mathcal{M}$, and during the payout phase the players will submit separate slates for s_1 and s_2 corresponding to the payouts for as_1c and $as_2 + b$.

In the case of our call option, the players might choose a value of $c = 10.00$, leading to $s_1 \in \{0, 150, 151, \ldots, 999\}$ and $s_2 \in \{0.00, 0.01, \ldots, 9.99\}$. They would thus have to sign only 1851 slates (851 slates for 851 possible values of s_1 and 1000 slates for 1000 possible values of s_2) instead of 850001, greatly reducing the amount of computation during setup of the instrument.

More generally, this technique can be applied to d-digit representations of x (i.e. $x = s_1c^{d-1} + s_2c^{d-2} + \cdots + s_d$) reducing the amount of off-chain computation by the players to $O(d|X|^{1/d})$. However, as each increase in d increases the amount of

[12]More generally, we only require that f be "piecewise composable." That is, for a fixed c, $f(cs_1 + s_2) = f_1(s_1) + f_2(s_2)$ for some f_1, f_2 for any pair (s_1, s_2). A linear function $f(x) = ax + b$ has this property, as $f(cs_1 + s_2) = f_1(cs_1) + f_2(s_2)$ for $f_1(x) = ax$ and $f_2(x) = f(x)$.

(expensive) on-chain computation that $\mathcal{M}$ needs to perform in the report and payout phases, we imagine that in practice $d \leq 2$ would suffice for almost all use cases. For better privacy, an actual implementation would use tags as outlined for SIMI-PPS in Section 5.2. As the extension is straightforward, we don't describe it here.

6.2 Extension to multiple players.

SIMI-PS can easily be extended to a set of $n \geq 2$ players $\mathcal{P}_0, \ldots, \mathcal{P}_{n-1}$ that are content with internal transparency, i.e., don't require confidentiality. In this case, all players would need to sign off on the instrument and payout slates to achieve theft-proofness. In turn, the participation of more players in a Mixicle contract results in stronger confidentiality against outsiders, since the active parties in any given bet enjoy privacy within a larger anonymity set.

We are currently exploring ways of achieving internal-player confidentiality within larger sets. In particular, it is possible to run *merged Mixicles*, in which multiple Mixicles execute in tandem. We are also exploring ways that external players can participate in the mixing operation of a Mixicle using ordinary transactions. Oracle nodes themselves might provide this kind of confidentiality as a service.

6.3 Tokens and other assets

The Mixicle smart contract is mostly agnostic to the implementation detail of what asset is being transferred by it. To use standard terms for token operations [87], as long as the asset offers balance-like functionality for checking ownership and transfer-like functionality for changing ownership, Mixicles can easily be modified to support it. A single Mixicle contract could also support multiple asset types simultaneously.

In particular, extending Mixicle to support ERC-20 tokens [87] and various derived token types such as ERC-777 and ERC-1155 tokens [28, 71] is straightforward. Even non-fungible tokens [34] (NFTs) could be easily supported and useful for, e.g., enabling a conditional sale of an NFT for currency with guaranteed delivery.

7 Related Work

Decentralized finance (DeFi). Most closely related to our work here on Mixicles is the Discreet Log scheme of Dryja [31], a pioneering work in oracle-based DeFi. The objective of that scheme is to support an oracle-based DeFi contract in Bitcoin that is indistinguishable from an ordinary Lightning Network transaction. In other words, the participation of an oracle in contract transactions is meant to be unobservable—as is the very fact that players are engaging in a contract.

The Discreet Log scheme, however, has somewhat different goals than Mixicles. It: (1) Presumes an oracle $\mathcal{O}$ that publicly broadcasts data off-chain or in the Bitcoin P2P network; (2) Assumes transactions using Schnorr signatures [74], which Bitcoin does not yet support; (3) Doesn't mix payments, i.e., conceal payments to players; (4) Aims to be "discreet," meaning that oracle participation is not visible; (5) Has strict requirements on users staying online; and (6) Requires settlement of the contract after each oracle report unless using Lightning channels.

Eskandari et al. [35] describe Velocity, a system initially implemented as an automated market maker for options contracts between ETH and another desired cryptocurrency, but without any form of confidentality.

Domain specific languages (DSLs) for financial instruments have a long history; see, e.g., [73] for a list of such DSLs. Inspired by [81] and the composable approach in [46], are Findel [12] and Marlowe [76], DSLs designed specifically for DeFi derivatives on blockchains, but not aiming at privacy. In a somewhat different but related vein, Clark et al. [27] proposed a design for decentralized prediction markets, a type of derivatives market, that has strongly influenced current designs.

Mixnets and mixers (tumblers). *Mix networks* or *mixes* (or *shuffles*) were introduced in a seminal paper by David Chaum [23] in 1981. Mix networks randomly permute messages to make sender identities private. Chaum's work on mixes, along with blind signatures for e-cash [22], which he also introduced, serve as a conceptual and technical foundation for cryptocurrency mixers.

Cryptocurrency mixers or tumblers, privacy privacy for transactions instead of messages. A number of such schemes, e.g., Coinjoin, Mixcoin, Tumblebit [14, 40, 60, 84], have been proposed for Bitcoin. A related approach is to integrate confidentiality features natively into a currency. Zcash [43, 72] provides better privacy than a mixer, in that its anonymity set is the entire user population (although see caveats in [49]). Monero [68, 86], a popular cryptocurrency, integrates mix-like functionality in which users can simulate transactions by other, non-participating users' to create ad hoc mix sets; Mimblewimble incorporates the same idea using a very different mechanism [45].

Tumblers may appear easier to implement in Ethereum in principle thanks to its support for smart contracts. Proposed constructions include MixEth [77], Miximus [8], and Möbius [65]. Apart from their high gas costs, a challenge discussed in Appendix A, is that fresh receiving addresses cannot anonymous withdraw funds because they do not have gas to do so. Zether [19] provides transaction privacy of a different sort than a mixer. In its basic form, it provides privacy for transaction amounts, but it can be extended to providing mixing functionality. Zether permits funds to be locked to a contract and can in principle be integrated fairly straightforwardly with Mixicles to make transaction amounts private while preserving auditability. We leave details for future work.

Our work here on Mixicles is the first to explore the fusion of oracles with concepts from tumblers and to highlight the rich functionality arising from a simple combination of the two.

Mixicles are not anonymous. They are not designed to conceal monetary flows, nor are they designed to conceal the identities of participating players as in anonymous cash systems, so they don't require an analog to coin tracing. Mixicles do make private who received what payouts, creating a need for the equivalent of owner tracing. The auditability of our Mixicles designs provides this feature, as well as visibility into Mixicle terms and execution.

Smart-contract confidentiality. As on-chain objects, smart contracts offer transparency, but cannot compute on confidential data. The only way to make smart contract data or logic privaet is to move some portion of it off chain. Several basic approaches have been proposed:

- *State channels* [30, 32, 61, 66, 70] execute transactions or smart contract logic off chain; they leverage an on-chain smart contract for setup and termination, and as an arbiter in case of faults. Their main drawback is that because updates happen off-chain, a monitoring capability (e.g., "Watchtowers" in the Lightning Network [69], a "Custodian" in Pisa [61], etc.) is needed to ensure against commitment of stale state on chain. We briefly discuss a state-channel variant on Mixicles in Section 5.6.

- *Arbitrum* [48] allows players to execute changes in a virtual machine (with potentially large state) with private, hashed representation on chain. While it has the attractive ability to handle very general logic, Arbitrum requires a system of verifiers to resolve disputes using a challenge-based scheme, and is in general somewhat complex. (Truebit [82] involves a similar idea for dispute resolution, but does not offer privacy.)

- *Trusted execution environments (TEEs)* such as Intel Software Guard eXtensions (SGX) (see, e.g., [63] for a recent overview) are a powerful, hardware-based tool for confidential and integrity-protected application execution. They can be used to construct high-trust oracles [91], to create knowledge marketplaces on blockchains [83], and even for wholesale execution of smart contracts [25], offering an inexpensive, high-performance alternative to Mixicle. Some users, however, may be reluctant to rely on TEEs given recently demonstrated vulnerabilities, e.g., [50, 56, 85, 90]—particularly for high-assurance applications.

  TEEs might, however, be deployed to support isolated components of Mixicles run by entities such as oracle operators whom users generally deem trustworthy. For example, a Mixicle might consume Town Crier data from some of the nodes in $\mathcal{O}$, achieve enhanced mixing (with an audit capability for regulatory compliance) using a TEE, etc.

- *Zero-knowledge proofs* can be used to prove off-chain state changes correct in a privacy-preserving way to an on-chain contract. This is the key idea in Hawk [52]. While Hawk is a rigorous, general framework, it has several drawbacks, including high computational cost, on-chain storage requirements, and implementation complexity, reliance on a third-party "manager" that has access to private data, and the need for trusted setup for the zk-SNARK used for each contract type.

- *Secure multiparty computation (MPC)* [36] enables computation on private data that is secret-shared across a committee of nodes. While there are a number of MPC software packages [5], and some proposals to use it for privacy in blockchain systems [94, 95], its complexity and computational cost have meant few practical applications of MPC in its most general form to date (see, e.g. [13]).[13] Specialized forms of MPC, however, e.g., two-player MPC (2PC) in [93], can be useful in realizing Mixicles.

Mixicles are designed to avoid the cost and complexity of these various approaches by exploiting the specific structure of basic oracle-driven DeFi instruments, especially those involving two players.

8 Conclusion

There is very little work to date reasoning about the security and confidentiality properties of smart contracts combined with oracles. In this initial paper, we've introduced Mixicles as a simple template for such combinations, and a useful springboard for the creation of a range of confidentiality-preserving DeFi instruments.

Mixicles leave room for a number of interesting avenues of future work, several of which we're exploring. Among these are:

- *Security proofs:* We've modeled and analyzed security informally in this work. Formal modeling and simulation-based proofs of security, as in [91], would be a useful next step.

- *Integration with complementary technologies:* Mixicles seem amenable to integration especially with privacy-enhancing technologies such as Zether [19] and Deco [93].

- *Harmonization with ISDA guidelines and documents:* The International Swaps and Derivatives Association (ISDA) provides standarized documents (e.g., the ISDA Master Agreement) used in many derivatives transactions today. Guidance is beginning to emerge on how smart contracts may fit into such existing practices [26, 80], and it is worth exploring how to harmonize Mixicles with it.

- *Enhanced mixing:* Merged Mixicles, as noted above, and the ability for external players to help enhance confidentiality offer possible avenues to achieving

[13]Indeed, Enigma [4], the company created to realize the MPC protocol of [95], has started embracing TEEs.

even stronger confidentiality than our constructions here. We are, for instance, exploring approaches by which oracle nodes can bolster Mixicle confidentiality.

- *Other DeFi instruments:* We've focused on binary options as a simple use case, and briefly discussed more complex instruments. We believe that a wide array of financial instruments can be realized in the Mixicle framework. A systematic exploration would be helpful.

We look forward to including these ideas and more in later versions of this paper and/or follow-up work, and welcome corrections and comments from the Chainlink community as we refine the early work presented here.

Disclaimer

This article is provided "as is" without any representations or warranties, express or implied. The authors make no representations or warranties in relation to the article or the information and materials provided within the article. Nothing in this article constitutes, or is meant to constitute, advice of any kind.

References

[1] NEO white paper. `https://docs.neo.org/docs/en-us/basic/whitepaper.html`.

[2] TLSnotary – a mechanism for independently audited https sessions. `https://tlsnotary.org/TLSNotary.pdf`, 10 Sept. 2014.

[3] APMEX live gold price charts & historical data. www.apmex.com, 2019. Referenced 2 Sept. 2019.

[4] Enigma Protocol website. `enigma.io`, Referenced Sept. 2019.

[5] Theory and practice of multi-party computation workshops: Mpc software. `http://www.multipartycomputation.com/mpc-software`, Referenced Sept. 2019.

[6] Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference* (2018), ACM, p. 30.

[7] Bank for International Settlements. BIS statistics: Global OTC derivatives market. `https://www.bis.org/statistics/d5_1.pdf`, Accessed Sept. 2019.

[8] barryWhiteHat. Miximus Github repository, 2018. Referenced Sept. 2019.

[9] Bellare, M., Boldyreva, A., Desai, A., and Pointcheval, D. Key-privacy in public-key encryption. In *International Conference on the Theory and Application of Cryptology and Information Security* (2001), Springer, pp. 566–582.

[10] Bentov, I., Ji, Y., Zhang, F., Li, Y., Zhao, X., Breidenbach, L., Daian, P., and Juels, A. Tesseract: Real-time cryptocurrency exchange using trusted hardware. *IACR Cryptology ePrint Archive, Report 2017/1153* (2017).

[11] Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Gollamudi, A., Gonthier, G., Kobeissi, N., Kulatova, N., Rastogi, A., Sibut-Pinote, T., Swamy, N., et al. Formal verification of smart contracts: Short paper. In *ACM Workshop on Programming Languages and Analysis for Security* (2016), pp. 91–96.

[12] Biryukov, A., Khovratovich, D., and Tikhomirov, S. Findel: Secure derivative contracts for Ethereum. In *International Conference on Financial Cryptography and Data Security* (2017), Springer, pp. 453–467.

[13] Bogetoft, P., Christensen, D. L., Damgård, I., Geisler, M., Jakobsen, T., Krøigaard, M., Nielsen, J. D., Nielsen, J. B., Nielsen, K., Pagter, J., et al. Secure multiparty computation goes live. In *International Conference on Financial Cryptography and Data Security* (2009), Springer, pp. 325–343.

[14] Bonneau, J., Narayanan, A., Miller, A., Clark, J., Kroll, J. A., and Felten, E. W. Mixcoin: Anonymity for bitcoin with accountable mixes. In *International Conference on Financial Cryptography and Data Security* (2014), Springer, pp. 486–504.

[15] Brasser, F., Müller, U., Dmitrienko, A., Kostiainen, K., Capkun, S., and Sadeghi, A.-R. Software grand exposure:{SGX} cache attacks are practical. In *11th {USENIX} Workshop on Offensive Technologies ({WOOT} 17)* (2017).

[16] Breidenbach, L., Daian, P., Tramèr, F., and Juels, A. Enter the Hydra: Towards principled bug bounties and exploit-resistant smart contracts. In *USENIX Security Symposium* (2018), pp. 1335–1352.

[17] Breidenbach, L., Kell, T., Manuskin, A., Detrio, C., Chin, D., Eskandari, S., Gosselin, S., and Doweck, Y. ProvEth. `https://github.com/lorenzb/proveth`.

[18] Brickell, E. F., Gemmell, P., and Kravitz, D. W. Trustee-based tracing extensions to anonymous cash and the making of anonymous change. In *SODA* (1995), vol. 95, pp. 457–466.

[19] Bünz, B., Agrawal, S., Zamani, M., and Boneh, D. Zether: Towards privacy in a smart contract world. *IACR Cryptology ePrint Archive, Report 2019/191* (2019).

[20] Buterin, V. EIP 86: Abstraction of transaction origin and signature. `https://eips.ethereum.org/EIPS/eip-86`, 2017.

[21] Buterin, V., et al. A next-generation smart contract and decentralized application platform. *white paper 3* (2014), 37.

[22] Chaum, D. Blind signatures for untraceable payments. In *Advances in cryptology* (1983), Springer, pp. 199–203.

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

[24] Chen, S., Wang, R., Wang, X., and Zhang, K. Side-channel leaks in web applications: A reality today, a challenge tomorrow. In *2010 IEEE Symposium on Security and Privacy* (2010), IEEE, pp. 191–206.

[25] Cheng, R., Zhang, F., Kos, J., He, W., Hynes, N., Johnson, N., Juels, A., Miller, A., and Song, D. Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts. In *IEEE Euro S & P* (2019).

[26] Clack, C. D., and McGonagle, C. Smart derivatives contracts: the isda master agreement and the automation of payments and deliveries. *arXiv preprint arXiv:1904.01461* (2019).

[27] Clark, J., Bonneau, J., Felten, E. W., Kroll, J. A., Miller, A., and Narayanan, A. On decentralizing prediction markets and order books. In *Workshop on the Economics of Information Security, State College, Pennsylvania* (2014).

[28] Dafflon, J., Baylina, J., and Shababi, T. EIP 777: ERC777 token standard. `https://eips.ethereum.org/EIPS/eip-777`, 2017.

[29] de Sande, A. V., and Schmidt, R. G. EIP 1077: Executable signed messages refunded by the contract. `https://eips.ethereum.org/EIPS/eip-1077`, 2018.

[30] Decker, C., and Wattenhofer, R. A fast and scalable payment network with Bitcoin duplex micropayment channels. In *Symposium on Self-Stabilizing Systems* (2015), Springer, pp. 3–18.

[31] Dryja, T. Discreet log contracts. `https://adiabat.github.io/dlc.pdf`, 2017.

[32] Dziembowski, S., Faust, S., and Hostáková, K. General state channel networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (2018), ACM, pp. 949–966.

[33] Ellis, S., Juels, A., and Nazarov, S. Chainlink: A decentralized oracle network. *Retrieved March 11* (2017), 2018.

[34] Entriken, W., Shirley, D., Evans, J., and Sachs, N. EIP 721: ERC-721 non-fungible token standard. `https://eips.ethereum.org/EIPS/eip-721`, 2018.

[35] Eskandari, S., Clark, J., Sundaresan, V., and Adham, M. On the feasibility of decentralized derivatives markets. In *International Conference on Financial Cryptography and Data Security* (2017), Springer, pp. 553–567.

[36] Goldreich, O., Micali, S., and Wigderson, A. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing* (1987), ACM, pp. 218–229.

[37] Goodman, L. M. Tezos — a self-amending crypto ledger. `https://tezos.com/static/white_paper-2dc8c02267a8fb86bd67a108199441bf.pdf`, 2014.

[38] Griffith, A. T. Ethereum meta transactions. `https://medium.com/@austin_48503/ethereum-meta-transactions-90ccf0859e84`, 2018.

[39] Hackett, R. Zcash discloses vulnerability that could have allowed 'infinite counterfeit' cryptocurrency.

[40] Heilman, E., Alshenibr, L., Baldimtsi, F., Scafuro, A., and Goldberg, S. Tumblebit: An untrusted bitcoin-compatible anonymous payment hub. In *Network and Distributed System Security Symposium* (2017).

[41] Higgins, S. Eu authorities shut down Bitcoin transaction mixer. *CoinDesk* (22 May 2019).

[42] Hildenbrandt, E., Saxena, M., Rodrigues, N., Zhu, X., Daian, P., Guth, D., Moore, B., Park, D., Zhang, Y., Stefanescu, A., et al. Kevm: A complete formal semantics of the ethereum virtual machine. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)* (2018), IEEE, pp. 204–217.

[43] Hopwood, D., Bowe, S., Hornby, T., and Wilcox, N. Zcash protocol specification. *Tech. rep. 2016–1.10. Zerocoin Electric Coin Company, Tech. Rep.* (2016).

[44] Jakobsson, M., and Yung, M. Distributed "magic ink" signatures. In *International Conference on the Theory and Applications of Cryptographic Techniques* (1997), Springer, pp. 450–464.

[45] Jedusor, T. E. Mimblewimble. `https://github.com/mimblewimble/docs/wiki/MimbleWimble-Origin`, 2016.

[46] Jones, S. P., and Eber, J.-M. How to write a financial contract.

[47] Juels, A., Kosba, A., and Shi, E. The ring of gyges: Investigating the future of criminal smart contracts. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (2016), ACM, pp. 283–295.

[48] Kalodner, H., Goldfeder, S., Chen, X., Weinberg, S. M., and Felten, E. W. Arbitrum: Scalable, private smart contracts. In *USENIX Security Symposium* (2018), pp. 1353–1370.

[49] Kappos, G., Yousaf, H., Maller, M., and Meiklejohn, S. An empirical analysis of anonymity in zcash. In *USENIX Security Symposium* (2018), pp. 463–477.

[50] Kocher, P., Horn, J., Fogh, A., , Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., Mangard, S., Prescher, T., Schwarz, M., and Yarom, Y. Spectre attacks: Exploiting speculative execution. In *40th IEEE Symposium on Security and Privacy (S&P'19)* (2019).

[51] Kocher, P. C. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Annual International Cryptology Conference* (1996), Springer, pp. 104–113.

[52] Kosba, A., Miller, A., Shi, E., Wen, Z., and Papamanthou, C. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE symposium on security and privacy (SP)* (2016), IEEE, pp. 839–858.

[53] KPMG Enterprise. Venture Pulse Q4 2018: Global analysis of venture funding. `https://assets.kpmg/content/dam/kpmg/xx/pdf/2019/01/kpmg-venture-pulse-q4-2018.pdf`, 2018.

[54] Libra Association Members. An introduction to Libra. `https://libra.org/en-US/wp-content/uploads/sites/23/2019/07/LibraWhitePaper_en_US-Rev0723.pdf`, 2019.

[55] Lind, J., Eyal, I., Kelbert, F., Naor, O., Pietzuch, P., and Sirer, E. G. Teechain: Scalable blockchain payments using trusted execution environments. *arXiv preprint arXiv:1707.05454* (2017).

[56] Lipp, M., Schwarz, M., Gruss, D., Prescher, T., Haas, W., Fogh, A., Horn, J., Mangard, S., Kocher, P., Genkin, D., Yarom, Y., and Hamburg, M. Meltdown: Reading kernel memory from user space. In *27th USENIX Security Symposium (USENIX Security 18)* (2018).

[57] Luu, L., Chu, D.-H., Olickel, H., Saxena, P., and Hobor, A. Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security* (2016), ACM, pp. 254–269.

[58] Marino, B., and Juels, A. Setting standards for altering and undoing smart contracts. In *International Symposium on Rules and Rule Markup Languages for the Semantic Web* (2016), Springer, pp. 151–166.

[59] Marsh, A. SocGen introduces Crypto to a $2 trillion market. *Bloomberg* (9 May 2019).

[60] Maxwell, G. Coinjoin: Bitcoin privacy for the real world. `https://bitcointalk.org/index.php?topic=279249.0`.

[61] McCorry, P., Bakshi, S., Bentov, I., Miller, A., and Meiklejohn, S. Pisa: Arbitration outsourcing for state channels. *IACR Cryptology ePrint Archive, Report 2018/582* (2018).

[62] McCorry, P., Hicks, A., and Meiklejohn, S. Smart contracts for bribing miners. In *International Conference on Financial Cryptography and Data Security* (2018), Springer, pp. 3–18.

[63] McKeen, F., Alexandrovich, I., Anati, I., Caspi, D., Johnson, S., Leslie-Hurd, R., and Rozas, C. Intel® software guard extensions (intel® sgx) support for dynamic memory management inside an enclave. In *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016* (2016), ACM, p. 10.

[64] Meiklejohn, S., and Mercer, R. Möbius: Trustless tumbling for transaction privacy. Cryptology ePrint Archive, Report 2017/881, 2017. `https://eprint.iacr.org/2017/881`.

[65] Meiklejohn, S., and Mercer, R. Möbius: Trustless tumbling for transaction privacy. *Proceedings on Privacy Enhancing Technologies 2018*, 2 (2018), 105–121.

[66] Miller, A., Bentov, I., Kumaresan, R., Cordi, C., and McCorry, P. Sprites and state channels: Payment networks that go faster than lightning. *arXiv preprint arXiv:1702.05812* (2017).

[67] Milutinovic, M., He, W., Wu, H., and Kanwal, M. Proof of luck: An efficient blockchain consensus protocol. In *proceedings of the 1st Workshop on System Software for Trusted Execution* (2016), ACM, p. 2.

[68] Noether, S., Mackenzie, A., et al. Ring confidential transactions. *Ledger 1* (2016), 1–18.

[69] Osuntokun, O. Hardening lightning. `https://cyber.stanford.edu/sites/default/files/hardening_lightning_updated.pdf.`, February 2018.

[70] Poon, J., and Dryja, T. The bitcoin lightning network: Scalable off-chain instant payments, 2016.

[71] Radomski, W., Cooke, A., Castonguay, P., Therien, J., Binet, E., and Sandford, R. EIP 1155: ERC-1155 multi token standard. `https://eips.ethereum.org/EIPS/eip-1155`, 2018.

[72] Sasson, E. B., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., and Virza, M. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy* (2014), IEEE, pp. 459–474.

[73] Schiller, T. Financial DSL listing. `http://www.dslfin.org/resources.html`, 2013.

[74] Schnorr, C.-P. Efficient signature generation by smart cards. *Journal of cryptology 4*, 3 (1991), 161–174.

[75] Securities Industry and Financial Markets Association (SIFMA). 2019 outlook: Trends in the capital markets. `https://www.sifma.org/wp-content/uploads/2018/12/2019-Outlook-FINAL.pdf`, 2018.

[76] Seijas, P. L., and Thompson, S. Marlowe: Financial contracts on blockchain. In *International Symposium on Leveraging Applications of Formal Methods* (2018), Springer, pp. 356–375.

[77] Seres, I. A., Nagy, D. A., Buckland, C., and Burcsi, P. Mixeth: efficient, trustless coin mixing service for ethereum. *IACR Cryptology ePrint Archive, Report 2019/341* (2019).

[78] Shlomovits, O., and Seres, I. A. Sharelock: Mixing for cryptocurrencies from multiparty ECDSA. Cryptology ePrint Archive, Report 2019/563, 2019. `https://eprint.iacr.org/2019/563`.

[79] Stadler, M., Piveteau, J.-M., and Camenisch, J. Fair blind signatures. In *International Conference on the Theory and Applications of Cryptographic Techniques* (1995), Springer, pp. 209–219.

[80] Swaps, I., and (IDSA), D. A. Legal guidelines for smart derivatives contracts: Introduction, Jan. 2019.

[81] Szabo, N. A formal language for analyzing contracts. *Satoshi Nakamoto Institute* (2002).

[82] Teutsch, J., and Reitwiessner, C. A scalable verification solution for blockchains. *URL: https://people. cs. uchicago. edu/teutsch/papers/truebit pdf* (2017).

[83] Tramer, F., Zhang, F., Lin, H., Hubaux, J.-P., Juels, A., and Shi, E. Sealed-glass proofs: Using transparent enclaves to prove and sell knowledge. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P)* (2017), IEEE, pp. 19–34.

[84] Tran, M., Luu, L., Kang, M. S., Bentov, I., and Saxena, P. Obscuro: A bitcoin mixer using trusted execution environments. In *Proceedings of the 34th Annual Computer Security Applications Conference* (2018), ACM, pp. 692–701.

[85] Van Bulck, J., Minkin, M., Weisse, O., Genkin, D., Kasikci, B., Piessens, F., Silberstein, M., Wenisch, T. F., Yarom, Y., and Strackx, R. Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution. In *27th USENIX Security Symposium (USENIX Security 18)* (2018), pp. 991–1008.

[86] Van Saberhagen, N. CryptoNote v 2.0. `https://bytecoin.org/old/whitepaper.pdf`, 2013.

[87] Vogelsteller, F., and Buterin, V. EIP 20: ERC-20 token standard. `https://eips.ethereum.org/EIPS/eip-20`, 2015.

[88] Wikipedia. Gold reserve — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Gold_reserve`, 2019. Accessed Sept. 2019.

[89] Wood, G., et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper 151*, 2014 (2014), 1–32.

[90] Xu, Y., Cui, W., and Peinado, M. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *2015 IEEE Symposium on Security and Privacy* (2015), IEEE, pp. 640–656.

[91] Zhang, F., Cecchetti, E., Croman, K., Juels, A., and Shi, E. Town Crier: An authenticated data feed for smart contracts. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security* (2016), ACM, pp. 270–282.

[92] Zhang, F., Eyal, I., Escriva, R., Juels, A., and Van Renesse, R. REM: Resource-efficient mining for blockchains. In *26th USENIX Security Symposium (USENIX Security 17)* (2017), pp. 1427–1444.

[93] Zhang, F., Maram, S. K. D., Malvai, J., Goldfeder, S., and Juels, A. Deco: Liberating web data using decentralized oracles for TLS, 2019. In submission.

[94] Zyskind, G., Nathan, O., and Pentland, A. Decentralizing privacy: Using blockchain to protect personal data. In *IEEE Symposium on Security and Privacy Workshops* (2015), pp. 180–184.

[95] Zyskind, G., Nathan, O., and Pentland, A. Enigma: Decentralized computation platform with guaranteed privacy. *arXiv preprint arXiv:1506.03471* (2015).

A Withdrawal Confidentiality in Ethereum

In Ethereum and related systems, all of our Mixicle constructions incur a subtle complication commonly connected with Ethereum tumblers, e.g., [77]. The problem is that if output addresses are to make withdrawals directly, they must be pre-funded in order to pay associated gas costs. Pre-funding undermines transaction-graph confidentiality; e.g., if Alice sends funds to $\mathcal{P}_0$ for gas, then $\mathcal{P}_0$ is linkable to Alice.

Several remedies are possible for Mixicles:

1. Have $\mathcal{O}$'s transaction trigger transfer of funds to output accounts.
2. $\mathcal{M}$ can refund and reward arbitrary players for pushing messages signed by Alice or Bob to it. This approach is commonly referred to as *executable signed actions* or *meta-transactions* [38]. The format for the messages to be relayed to the smart contract has been standardized in EIP 1077 [29].
3. Have one of Alice or Bob, pre-selected uniformly at random, initiate disbursement of funds; to incentivize compliance, ensure that the payout slates unconditionally include funds for both players.
4. Use the not-yet-implemented `PAYGAS` opcode in Ethereum EIP86 [20] so that $\mathcal{M}$ can pay the gas cost of the transaction.
5. Pre-funding output accounts in a privacy-preserving way.

At the time of writing, (1), (2), and (3) are the only practical options.

B Advanced Options

B.1 Addressing timing side-channels

Our constructions make oracle queries private with respect to external observers. They can, however, leak data through a *timing side channel.* In other words, the *timing* of operations in a Mixicle can leak information, even if details of Mixicle are not explicitly revealed.[14]

If $\mathcal{O}$ sends $\mathsf{switch}(x)$ to $\mathcal{M}$ immediately upon determining its value, then external observers learn the time at which x itself was determined. For example, if Alice and Bob are betting on the outcome of the women's 100m sprint at the 2024 Olympics, and $\mathcal{M}$ receives data from $\mathcal{O}$ as the winner crosses the finish line, external observers would obtain evidence that this particular Olympic event was the basis for the financial instrument.

[14]Timing side channels were introduced in [51]. For an instructive overview of web-application side-channel vulnerabilities, including timing, see [24].

Even the timeout t_{report} on $\mathcal{O}$ reports leaks some information about switch. For example, if t_{report} *preceeds* the time of the women's 100m sprint at the 2024 Olympics, then that event can be ruled out as the object of switch.

We propose a couple of ways of addressing timing side channels in Mixicles:

1. *Predetermined, randomized delay.* A (randomized) delay Δ can be imposed on the oracle's transmission of $\mathsf{switch}(x)$, i.e., the report is delivered at time Δ after it becomes available. This delay Δ should be reflected in terms to make oracle compliance auditable.

   Of course, a delay in the delivery of $\mathsf{switch}(x)$ also delays the ability of the players to Mixicle to withdraw their funds. Immediate withdrawal on the resolution of switch, however, would itself leak the same timing information as immediate oracle input. Another drawback of imposing a predetermined delay Δ is its flexibility: Players cannot change it should it later prove undesirable.

2. *Switch authorizations.* A more flexible approach is to use what we call *switch authorizations.* A switch authorization is a signed report $\Sigma = (\mathsf{switch}(x), \sigma_{\mathcal{O}}[\mathsf{switch}(x)])$ that $\mathcal{M}$ consumes in lieu of direct input from $\mathcal{O}$.

   To ensure that which Mixicle the switch authorization Σ corresponds to is private, Σ can be: (1) Delivered to a *pool*, e.g., a shared contract $\mathcal{S}$, to which switch authorizations of multiple Mixicles are sent; and (2) Encrypted under the keys of the players to the Mixicle using a cipher with *key privacy* (e.g., El Gamal) [9]; i.e., the ciphertext should not reveal the associated public key, thereby compromising the privacy of the receiver. $\mathcal{S}$ may be thought of loosely here as a form of (asynchronous) mixer for Mixicle inputs.

   Of course, the plaintext Σ must be delivered to $\mathcal{M}$ by one of the participating players. Only a player that receives funds is incentivized to perform this delivery, so the confidentiality concerns discussed in Appendix A must be addressed.

   Other approaches are also possible, e.g., an "optimistic" one in which the default is off-chain delivery of Σ. If delivery has not occurred off-chain by the timeout t_{report}, then a player can post an encrypted request to $\mathcal{S}$ requesting delivery, i.e., requiring on-chain compliance with terms. Failure by $\mathcal{O}$ is then an auditable event, as the encrypted request and $\mathcal{O}$'s signature on terms can be shown to an auditor. Note that in the optimistic case here, the deadline t_{report} is never revealed.

B.2 Efficient payout slates with perfect payout privacy

Recall that an observer—either an external player or $\mathcal{O}$—can see the total amount of money $\$\pi$ disbursed in a payout slate π as well as (upon settlement) the individual payouts $\{\pi[i] = (\mathcal{P}_i, \$p_i)\}$. We showed in Section 4.2 how to structure the payouts in

π to confer privacy on the amounts paid to individual players. Our technique there, however, assumed $\$\pi = \$2^\ell - 1$ for some positive integer ℓ.

We now show how to accomplish perfect payout privacy for general $\$\pi$. As with out construction above, we assume \$1 granularity, and achieve $n = \lceil \log \$\pi \rceil$.

More formally, we want a multiset $D = \{\$d_i\}_{i=1}^n$, such that

$$\$\pi = \sum_{i=1}^{n} \$d_i$$

and

$$\forall \$a \in [\$\pi] . \exists I \subseteq [n] . \$a = \sum_{i \in I} \$d_i$$

i.e., D is a multiset of "denominations" that should sum to the total $\$\pi$ to be disbursed by the contract while also allowing us to pay out any amount up to that total to one player (by giving her all slate items in I) and giving the remainder to the other player (by giving her all slate items in $\bar{I}$).

We can reformulate the second condition to an equivalent one,

$$\forall \$a \in \left[\left\lfloor \frac{\$\pi}{2} \right\rfloor \right] . \exists I \subseteq [n] . \$a = \sum_{i \in I} \$d_i$$

since any amount $\$a > \left\lfloor \frac{\$\pi}{2} \right\rfloor$ can be assembled by looking at the complement index set $\bar{I}$ of $\$\pi - \$a$.

This leads to a recursive formula for finding $D_{\$\pi}$ for any $\$\pi$,

$$D_{\$\pi} = \begin{cases} \{1\} & \text{for } \$\pi = 1 \\ D_{\lfloor \$\pi/2 \rfloor} \cup \{\$\pi - \lfloor \$\pi/2 \rfloor\} & \text{for } \$\pi > 1 \end{cases},$$

which can be proved to satisfy our two criteria by straightforward induction. Notably, the number of elements $|D_{\$\pi}|$ is in $O(\log \$\pi)$, allowing us to construct efficient slates even for large values of $\$\pi$. For example, for $\$\pi = 1,000,000$, we end up with a slate with only 20 elements:

$$\begin{aligned} D_{10^6} = \{&1, 2, 4, 8, 15, 31, 61, 122, 244, 488, 977, 1953, 3906, \\ &7813, 15625, 31250, 62500, 125000, 250000, 500000\} \end{aligned}$$

C Trusted Execution Environments (TEEs)

If Alice and Bob are willing to trust a *trusted execution environment* (TEE) such as Intel SGX to enforce correctness and confidentiality, it is easy and simple to construct a powerful Mixicle. Indeed, TEEs have been proposed for a number of blockchain protocols; see, e.g., [10, 25, 47, 55, 67, 91, 92].

Consider the following example, with SIMI modified so that $\mathcal{O}$ runs in a TEE (enclave). Alice and Bob send input funds to $\mathcal{M}$. They send $\mathcal{P}_0$ and $\mathcal{P}_1$, along with a specification of switch, to $\mathcal{O}$. $\mathcal{O}$ then evaluates $\mathsf{switch}(x)$ off chain, and directs $\mathcal{M}$ to send input funds to $\mathcal{P}_{\mathsf{switch}(x)}$. An outside observer only sees $\mathcal{O}$ direct funds to a fresh, random address—nothing more.

Alternatively, $\mathcal{O}$ could just take Alice and Bob's funds in custody on chain and execute $\mathcal{M}$ entirely off chain, subsequently routing funds to the right player on chain. In this case, $\mathcal{O}$ is implementing a *private contract*.

In both cases, realizing $\mathcal{O}$ as a TEE ensures the same confidentiality and integrity properties as SIMI, *plus* privacy for switch (although the operator of $\mathcal{O}$ could see which servers $\mathcal{O}$ interacts with). The first variant could be implemented using Chainlink / IC3's Town Crier system [91]. The second, private contract variant is a clear use case for Ekiden [25], which could execute the contract entirely off-chain in a trusted manner.

Trusted hardware is imperfect, of course. Serious attacks against SGX have come to light involving controlled channels, shared cache, and speculative execution, e.g., [15, 50, 56, 85, 90]. Some users may therefore feel uncomfortable using TEEs. It is for this reason that we designed Mixicles *without* a reliance on TEEs.

An interesting direction to consider is use of TEEs for *correctness*, but not *confidentiality*, as in the Sealed-Glass Proof model of [83].