Platypus: Offchain Protocol Without Synchrony

Alejandro Ranchal-Pedrosa
University of Sydney
Sydney, Australia
alejandro.ranchalpedrosa@sydney.edu.au

Vincent Gramoli
University of Sydney and Data61-CSIRO
Sydney, Australia
vincent.gramoli@sydney.edu.au

***Abstract*—Offchain protocols aim at bypassing the scalability and privacy limitations of classic blockchains by allowing a subset of participants to execute multiple transactions outside the blockchain. While existing solutions like payment networks and factories depend on a complex routing protocol, other solutions simply require participants to build a *childchain*, a secondary blockchain where their transactions are privately executed. Unfortunately, all childchain solutions assume either synchrony or a trusted execution environment. In this paper we present Platypus, an offchain protocol that requires neither synchony nor a trusted execution environment. Relieving the need for a trusted execution environment allows Platypus to ensure privacy without trusting a central authority, like Intel, that manufactures dedicated hardware chipset, like SGX. Relieving the need for synchrony means that no attacker can steal coins by leveraging clock drifts or message delays to lure timelocks. In order to prove our algorithm correct, we formalize the chilchain problem as a Byzantine variant of the classic Atomic Commit problem, where closing an offchain protocol is equivalent to committing the whole set of payments previously recorded on the childchain "atomically" on the main chain. Platypus is resilience optimal and we explain how to generalize it to crosschain payments.**

***Index Terms*—Byzantine atomic commit, scalability, privacy, childchain, blockchain**

I. Introduction

One of the most important challenges of blockchains is scalability. In fact, most blockchains consume more resources without offering better performance as the number of participants increases. Although some research results demonstrated that blockchain performance can scale with the number of participants [7], these rare solutions do not have other appealing properties, like privacy, built in. As a result, blockchain extensions that offer scalability and privacy have been put forward, in what is known as *Offchain protocols*. Examples of these protocols are state and payment *channels* [25] in which two parties can perform several offchain payments with one another; *channel networks* [25] that allow users to relay payments in a network of channels; *channel factories* [27] that open multiple channels in one transaction, saving storage and fees; and *childchains* [24] which are secondary blockchains pegged to the existing, so called "parent", blockchain.

By relying on offchain computation, all these protocols avoid communicating and/or storing some information directly in the blockchain—hence bypassing the performance bottleneck of the blockchain but also limiting transparency of selected transactions to ensure privacy. Channel networks and channel factories offer private and fast payments, they can only perform payments if users have an existing route of channels with one another. As a result their scalability and privacy are actually subject to proper handling of the network topology and vulnerable to routing attacks [16].

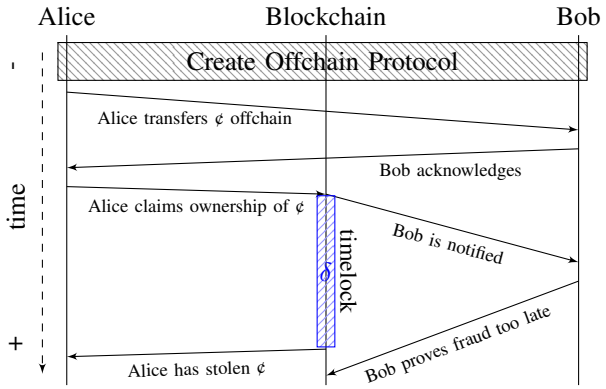

Fig. 1. Alice can steal Bob's coin if Bob messages are delayed such that Bob's reply takes longer than the timelock δ.

There is, therefore, great interest in designing proper childchain protocols that allow blockchains to host the creation and destruction of other smaller blockchains that depend on them. Unfortunately, as far as we know all childchains [2], [12] use timelocks that only work under the assumption that the communication is *synchronous*, in that every message gets delivered in less than a known bounded amount of time [9]. This assumption is easily violated in large networks like the internet, due to either natural disasters or human misconfiguration of BGP tables for example. But more dramatically, assuming synchrony exposes childchains to various attacks, like Denial-of-Service or Man-in-the-Middle, that are common practice to double spend [10].

To illustrate the problem of chilchains, consider an execution using timelocks illustrated in Figure 1 in which time increases from top to bottom. First, Alice transfers ¢ coins to Bob offchain before Bob acknowledges the transfer. Bob can then take actions in response to this transfer thinking, wrongly, that he will have sufficient time to prove the fraud if Alice tries to claim back the coins. Let us consider that Alice is Byzantine (or malicious) and claims back the ownership of the coins, which triggers a timelock, a safe guard delay during which the coins are locked to give an opportunity to other participants to prove fraudulent activity before the coins are transferred back. As part of the protocol, Bob gets notified

but due to an unforeseen delay, does not manage to prove the fraud before the end of the timelock. Its ¢ coins are thus stolen.

In this paper, we propose *Platypus*, the first offchain protocol for childchains that does not assume synchrony. To this end, we formalize offchain protocols as a Byzantine fault tolerant atomic commit of multiple transfers. Platypus exploits *partial synchrony* [9]—also called eventual synchrony—where messages take unknown amounts of time to be delivered. We prove the correctness and resilience optimality of Platypus and discuss applications to crosschain payments. Finally, we show that the time, message and communication complexities of Platypus are lower or comparable to consensus algorithms.

The rest of this document is structured as follows: Section II provides some background and preliminary definitions, Section III introduces our model and Section IV presents the Platypus protocol. We prove Platypus correct in Section V. We analyse the complexity of Platypus in Section VI. Section VII discusses applications of Platypus to crosschain payments. Section VIII presents the related work and we conclude in Section IX.

II. Preliminaries

In this section we discuss the background and introduce important notations.

• *Blockchain.* Inspired by [12], we refer to a blockchain $\Omega = \langle b_i \rangle$ as a distributed ledger that builds upon a consensus protocol in order to add *blocks* b_i. We denote $\Omega[i]$ as the i^{th} block of Ω, with $\Omega[-i]$ being the i^{th} latest block of Ω. Transactions are added to a block that is then written in Ω. The set of processes V that agree on the transactions to be written in the next block are the *validators*. As we assume the presence of a consensus protocol, we consider that the blockchain cannot fork due to a disagreement, we call it *unforkable*.

• *Transactions.* Similar to the model of [13], a transaction is a tuple $tx = \langle I, O \rangle$ where I is a list of inputs and O a list of outputs. Outputs are stored in an Unspent Transaction Output (UTXO) pool until a transaction that consumes it as one of its inputs gets written in Ω. We model the outputs as $o_i = \langle s_i, ¢_{o_i} \rangle$ where the set $s_i = \{(p_i, conds_{p_i})\}$ defines the conditions $conds_{p_i}$ for the process p_i to spend the coin $¢_{o_i}$ ($¢_{o_i} \geq 0$). In order to spend a coin, the associated conditions $conds_{p_i}$ must be fulfilled so that only one process, among multiple candidate ones, can spend this coin.

• *Ownership.* We say that process p_i owns coin $¢_i$ if there exists a list of conditions $conds_{p_i}$ such that p_i can spend $¢_i$. As such, let $\mathbb{C}$ be the set of coins, $\mathcal{T}$ the set of discrete timeslots (such as blockheight), and P the set of processes, then ownership is a function $\varphi : \mathbb{C} \times \mathcal{T} \to P$ that takes a coin and returns its owner at a particular time.

• *Transferring coins.* A transaction may transfer one or more coins. We refer to p_i transferring a coin $¢_i$ to p_j if a process p_i spends it to p_j. We can define a transfer of a coin as a change of ownership. That is, let a, $b \in P$ and let $¢_i \in \mathbb{C}$, $t_j \in \mathcal{T}$, such that $\varphi(¢_i, t_j) = a$, then the transfer relation TR to b is such that $a \; TR_{t_{j+1}, ¢_i} \; b \iff \varphi(¢_i, t_{j+1}) = b$. Notice we can define the transitive closure of the transfer operation as follows:

$$TR^+ = \left\{ \begin{array}{c} (a,b) \in P^2 : \exists ¢_i \; s.t. \; \varphi(¢_i, t_j) = a \textbf{ and } \varphi(¢_i, t_k) = b, \\ \textbf{for some } t_j, t_k \in \mathcal{T}, \; j < k. \end{array} \right\} \quad (1)$$

III. Model

In this section, we present our model and our assumptions.

• *Partial synchrony and failures.* Our model relies on a partially synchronous network [9], i.e. a network in which each message has an unknown communication bound. In other words, there exists an unknown time where the network stabilizes and after which every messages is delivered in a bounded amount of time. We also assume that strictly less than $n/3$ processes participating in an offchain protocol are *Byzantine* in that they may fail arbitrarily and that other processes are *correct* (as we detail in the adversary model below).

• *Accounts.* We define an account a as an instance of only one process p_i, $\rho(a) = p_i$, where $\rho(a)$ is a function that returns the process that controls account a. An account belongs to a particular blockchain, one account is controlled by only one process, but one process can have multiple accounts, either in the same or in different blockchains.

• *Threshold signatures.* Our model requires accounts to authenticate with a cryptographic primitive enabling non-interactive aggregation, such as those of [3], [12]. For simplicity and without loss of generality we assume that accounts are not reusable. In particular, the same coins should not go back to the same process in the same account, to prevent a variant of the ABA problem (see Section VII). In the remainder, we abuse the term process as an account that the process owns, unless stated otherwise.

• *Minimal transfers.* Given a sequence $seq = \{u_j \; TR_{t_{j+1}, ¢_i} \; u_{j+1}\}_{j=c}^{d-1}$ of transfers over some timerange $[t_c, t_d]$, between creation time t_c and destruction time t_d, for coin $¢_i$, we refer to the minimal transfer as the single transfer $u_c \; TR_{¢_i} u_d$, which is always an element of the transitive closure. For a set of operations defined over all coins within a timerange $[t_c, t_d]$, we denote the minimal transfer set TR^- as the set of all minimal transfers, which is at least a set of idempotent transfers of the form $a \; TR \; a$.

• *Offchain problem.* Given a blockchain Ω of P processes, the offchain protocol consists of executing a sequence seq of transfers "off chain". First processes $Q \subsetneq P$ must create an offchain protocol Γ by writing a transaction in the original chain Ω—effectively depositing funds from Ω into Γ. Then they transfer coins "off chain" among themselves using Γ. Finally they can destroy this protocol Γ. To this end, the offchain protocol consists of at least two main procedures, *creation* and *bulk close*. (We will explain later how the participation in Γ is made dynamic using splice in and splice out procedures to accept new participants and for existing participants to leave Γ, respectively). After a series of transfers in Γ, processes can propose to bulk close it by *proposing COMMIT*. Processes

decide to COMMIT in that they effectively agree to accept these transfers and to close and destroy Γ or *decide to ABORT* in that they disagree with the transfers and refuse to close Γ. Hence, a protocol solving the *Offchain problem* must satisfy the following properties:

- Termination: every correct process decides COMMIT or ABORT on some sequence of transfers seq for which some process proposed COMMIT.
- Agreement: no correct process decides COMMIT on two different sequences seq and seq'.
- ABORT-Validity: if a correct process proposes ABORT for a sequence seq, then all correct processes decide ABORT for this sequence seq.
- COMMIT-Validity: if no correct process proposes ABORT for sequence seq for which some process proposed COMMIT, then all correct processes decide COMMIT for sequence seq.

Notice that, in our definition, aborting is implicit and proposing ABORT is not an input of our algorithm as we will see in Algorithm 3. In particular, COMMIT-Validity can be ensured by requiring a process to provide a valid Proof-of-Fraud (PoF) when proposing to abort, the invalidity of the PoF allows correct processes to ignore the ABORT proposal and its validity guarantees that all correct will observe this PoF. Finally note that our termination does not imply that the offchain protocol gets closed. Instead, it means that all correct processes decide either COMMIT and closes the protocol or ABORT and not closing the protocol. This is not a problem since, as we will explain in Algorithm 5, any correct process can cash out the coins that it knows it owns at any moment.

In order to achieve privacy, we need another property stating that some decisions of the offchain protocol do not have to be written in the blockchain:

- COMMIT-Privacy/Lightness: If correct processes decide COMMIT on a sequence seq of transfer operations made in Γ between t_c and t_d, then $\forall p \in P \backslash Q$, p only learns/stores TR^{-}, the minimal transfer set of seq.

• *Childchain.* A *childchain* Ψ is a particular class of offchain protocol in that it is a blockchain Ψ that is created by another blockchain Ω, known as its *parentchain*, and that implements an Offchain protocol Γ.

• *Adversary model.* We consider an adversary F such that:

- F can control the network to read or delay messages, but not to drop them.
- F can take full control and corrupt a coalition of f processes, learning its entire state (stored messages, signatures, etc.). It takes control of receiving and sending all their messages. This adversary can also guess in advance the estimate value of any correct process in any round. Furthermore, it delivers the messages from correct nodes instantly, and its messages are delivered instantly by any correct nodes.
- F cannot forge signatures of processes outside the coalition f.
- We define t_0 and t_1 two thresholds for Byzantine behavior. That is, the coalition must be such that $f \leq t_0$ and $f \leq t_1$.

IV. Secure Childchains Without Synchrony

In this section we present Platypus, a novel childchain protocol that solves the offchain problem without assuming synchrony. Platypus consists of both an offchain protocol and a childchain that are denoted respectively Γ and Ψ in the remainder of the paper. Given a parentchain Ω, processes can use the protocol Γ by depositing funds from Ω to Ψ, that effectively creates the childchain. Then transfers can be done directly on Ψ "off chain" before the bulk close happen.

The parentchain Ω and childchain Ψ have a set P_Ω of $|P_\Omega| = n_p$ users and P_Ψ of $|P_\Psi| = m_p$ users, respectively, with a set $V_\Omega \subseteq P_\Omega$ of $|V_\Omega| = n_v$ validators and a set $V_\Psi \subseteq P_\Psi \subseteq P_\Omega$ of $|V_\Psi| = m_v$ validators, respectively. Note that m_v is the number of all processes joining the Platypus protocol. As mentioned before, however, we assume that at most $t_1 = \lceil m_v/3 \rceil - 1$ among them are Byzantine. Although we do not provide an implementation of the blockchain Ψ we assume that Ψ is secure (i.e. it uses deterministic consensus to not fork): a blockchain assuming partial synchrony and $\lceil m_v/3 \rceil - 1$ Byzantine processes among m_v processes like Red Belly [6], [7] can be used here.

A. Overview

Γ is depicted in three main procedures: a creation (Alg. 1), a bulk close (Alg. 2) and an abort (Alg. 3). (Splice in and splice out procedures are deferred to Section VII). Processes can ABORT or COMMIT sequences of transfers done in Ψ. In particular, a process proposes ABORT by creating an abort transaction (and sharing it) in line 7 of Alg. 3 and proposes a COMMIT at line 9 of Alg. 2. A process decides COMMIT at line 10 (Alg. 2) only after m_0 processes propose COMMIT and decides ABORT at line 11 (Alg. 2) only when there exists a valid abort transaction.

B. Creating a Platypus Chain

Users can create a Platypus chain by publishing a transaction on Ω. After that transaction is finalized, the funds referred to in this transaction are locked and ready to be used by the Γ. In general, a Platypus creation transaction (tx_{plcr}) is a transaction that:

- Has a new Platypus id ($plid$) that uniquely identifies it.
- Specifies a consensus protocol for the Platypus Blockchain to decide on a new block. W.l.o.g., we assume DBFT [6] to be the default protocol.
- Specifies a number $m_0 > f$ of validators required to create Ψ. For simplicity and to match with the optimal result (see Theorem 2), we choose $m_0 = \lfloor 2m_v/3 \rfloor + 1$.
- Defines a new function $\mathsf{abort}(...)$ that specifies when a user can decide ABORT on the protocol (such as a Platypus bulk close transaction being aborted).
- Specifies a set of processes and their balances that go in the Platypus Blockchain through this transaction.
- Once written in Ω, the funds can only be spent in Ψ.

Algorithm 1 shows the protocol to create a Platypus chain. The call to $\mathsf{num\_signers}(tx)$ returns the amount of signers of tx, while the call to $\mathsf{verify}(\mathsf{tx}, \{\mathsf{msg}\})$ verifies the validity

Algorithm 1 Platypus creation procedure

▷ State of the algorithm
Ω, the parentchain
Γ, the Platypus protocol
P_Ω, the set of processes in the parentchain
$P_\Psi \leftarrow \perp$, the set of processes in the Platypus chain
$V_\Psi \leftarrow \perp$, the set of validators in the Platypus chain
m_v, the amount of validators required in Ψ
$\mathbb{C}_i$, coins that belong to process p_i
job_i, boolean defining if p_i is VALIDATOR or just USER
$plid$, the Platypus chain identifier
$msg_i = \langle \mathbb{C}_i, plid, job_i\, \sigma_i \rangle$, signed message to join.
σ_i, signature of msg_i by p_i
$tx_{plcr} \leftarrow \perp$, the Platypus creation transaction

▷ PHASE 1: process p_0 initiates request
1: $msg_0 \leftarrow \mathsf{sign}(\langle \mathbb{C}_0, plid, \mathsf{job}_0 \rangle)$
2: $\mathsf{multicast}(msg_0)$ to P_Ω

▷ PHASE 2: Rest of processes who want to join reply
3: **when** msg_0 is received from p_0
4: $msg_i \leftarrow \mathsf{sign}(\langle \mathbb{C}_i, plid, \mathsf{job}_i \rangle)$
5: $\mathsf{multicast}(msg_i)$ to P_Ω

▷ PHASE 3: Validator $p_i \in V_\Psi$ gathers enough validators
6: **when** msg_j is received from p_j **and** $p_j \notin P_\Psi$
7: $\{P_\Psi, \mathbb{C}_{P_\Psi}\} \leftarrow \{P_\Psi \cup \{p_j\},\ \mathbb{C}_{P_\Psi} \cup msg_j.\mathbb{C}_j\}$
8: **if** $(msg_j.job_j = \mathsf{VALIDATOR}$ **and** $p_j \notin V_\Psi)$ **then**
9: $\{V_\Psi, \mathbb{C}_{V_\Psi}\} \leftarrow \{V_\Psi \cup \{p_j\},\ \mathbb{C}_{V_\Psi} \cup msg_j.\mathbb{C}_j\}$
10: **if** $(|V_\Psi| = m_v)$ **then** ▷ *Enough validators to start transaction*
11: $tx_{plcr} \leftarrow \mathsf{createPlatypusTx}(\mathbb{C}_{P_\Psi}, \mathbb{C}_{V_\Psi}, plid)$
12: $tx_{plcr} \leftarrow \mathsf{sign_i}(tx_{plcr})$
13: $\mathsf{multicast}(tx_{plcr}, \{msg_k\}_{p_k \in P_\Psi})$ to V_Ψ

▷ PHASE 4: $p_i \in V_\Psi$ signs and broadcasts until it gets enough signatures
14: **when** $(tx_{plcr}, \{msg_j\}_{p_j \in P_\Psi})$ is received **and not** $\mathsf{is\_written}(\Omega, tx_{plcr}, plid)$ ▷ *if tx_{plcr} with $plid$ not written in Ω*
15: **if** $(\mathsf{verify}(tx_{plcr}, \{msg_j\}))$ **then** $tx_{plcr} \leftarrow \mathsf{sign_i}(tx_{plcr})$
16: **if** $(\mathsf{num\_signers}(tx_{plcr}) < \lfloor 2m_v/3 \rfloor + 1)$ **then**
17: $\mathsf{multicast}(tx_{plcr}, \{msg_j\})$ to V_Ψ
18: **else** $\Gamma.\mathsf{send}(\Omega, tx_{plcr})$ ▷ *enough signatures*

of the transaction and signed messages. We define two main interactions of the Platypus protocol with both the child-chain and the parentchain: sending transactions and reading transactions. The Platypus protocol Γ sends transactions to Ω or Ψ by invoking $\mathsf{send}(\{\Omega, \Psi\}, \mathsf{tx})$ and $\mathsf{acsend}(\{\Omega, \Psi\}, \mathsf{tx})$—standing for "atomic commit send". In the former, the function returns once the transaction is written in the corresponding blockchain or a transaction that spent the same funds has been written (meaning this transaction became invalid), while the latter returns ABORT or COMMIT and the respectively written transaction in a response message. This response is received by all validators as it is a result of the Platypus Blockchain. Reading transactions is performed by the call to $\mathsf{is\_written}(\{\Omega, \Psi\}, \mathsf{tx})$ that returns True or False depending on whether the transaction was written or not in the Blockchain. Each of the messages is signed, to prevent Byzantine nodes from adding third parties without their agreement.

Algorithm 2 Platypus bulk close procedure

▷ State of the algorithm
Ω, Ψ, Γ, the Blockchain, Platypus Blockchain and protocol
P_Ψ,V_Ψ, the set of processes and validators in Ψ
$\mathbb{C}_i$, the coins that belong to process p_i
$tx_{plcl} \leftarrow \perp$

▷ PHASE 1: Some process p_0 creates and broadcasts
1: $tx_{plcl} \leftarrow \mathsf{createBulkCloseTx}(\mathbb{C}_{P_\Psi})$
2: $tx_{plcl} \leftarrow \mathsf{sign_i}(tx_{plcl})$
3: $\mathsf{multicast}(\mathsf{tx_{plcl}})$ to V_Ψ

▷ PHASE 2: $p_i \in V_\Psi$ signs and broadcasts transaction
4: **when** tx_{plcl} is received **and not** $\mathsf{is\_written}(\Psi, tx_{plcl})$
5: $\mathsf{verify}(tx_{plcl})$
6: $tx_{plcl} \leftarrow \mathsf{sign_i}(tx_{plcl})$
7: **if** $(\mathsf{num\_signers}(tx_{plcl}) < \lfloor 2|V_\Psi|/3 \rfloor + 1)$ **then**
8: $\mathsf{multicast}(tx_{plcl})$ to V_Ψ
9: **else** $r \leftarrow \Gamma.\mathsf{acsend}(\Psi, tx_{plcl})$ ▷ *Get back tx_{plcl}, or tx_{Abort}*

▷ PHASE 3: $\Gamma.\mathsf{acsend}(\Psi, tx_{plcl})$ generates a response, any p_i can send to Ω
when r is received
10: **if** $(r.type = \mathsf{ABORT})$ **then** $\Gamma.\mathsf{send}(\Omega, r.tx_{Abort})$
11: **else if** $(r.type = \mathsf{COMMIT})$ **then** $\Gamma.\mathsf{send}(\Omega, r.tx_{plcl})$

C. Closing a Platypus Chain

A Platypus bulk close transaction splices all funds out of the Platypus Blockchain without compromising its security (agreement), and without requiring all validators to join together in its destruction (termination). It is still a normal transaction in the Platypus blockchain, meaning that it requires enough validators m_0 to agree with writing it in Ψ. Algorithm 2 shows the protocol to bulk close a Platypus chain. A Platypus bulk close transaction signed by some processes returns back the updated balances of all processes in the parentchain Ω, unless it is aborted. Once written in both Ψ and Ω, the coins can be spent only on Ω.

D. Aborting a Closing Attempt

A Platypus bulk close transaction with insufficient signatures can either be a valid, ongoing Platypus bulk close, or an attempt to commit fraud. To prevent this, and guarantee termination and ABORT-validity, we introduce the abort transaction.

A transaction may be invalid if it spends a coin formerly owned by a user, but that was transferred to another user later on in Ψ. The abort function runs for every Platypus bulk close transaction received that is not valid, i.e. that spends some input already spent. If the transaction is not valid due to signatures not matching, then it will not be written in the parentchain, so the abort function ignores this case.

Therefore, a user can see that a transaction tx is not valid if an old owner claims ownership of a spent coin in tx, as checked by $\mathsf{coins\_spent}(...)$, shown in Algorithm 3. Notice that, while a COMMIT requires m_0 validators to commit to the transaction (such as a Platypus bulk close transaction), any process $p \in P_\Psi$ can create a valid abort transaction. The call to $\mathsf{extract\_spent}(tx)$ returns the coins that were spent. The call to $\mathsf{get\_block\_min\_blockheight}(\mathbb{C}_S)$ returns the

Algorithm 3 Abort procedure

▷ State of the algorithm
Ω, Ψ, Γ, the Blockchain, Platypus Blockchain and protocol.
$\mathbb{C}_S \leftarrow \perp$, the subset of spent coins from $\mathbb{C}$
$b_p \leftarrow \perp$, integer s.t. $\Psi[b_p]$ proves some coin was spent
$vPoF \leftarrow \perp$, Proofs-of-Fraud of validators
$tx_{abort} \leftarrow \perp$

1: **function** abort(tx_{plcl})
2: $\mathbb{C}_S \leftarrow$ extract_spent(tx_{plcl})
3: $b_p \leftarrow$ get_block_min_blockheight($\mathbb{C}_S$)
4: $vPoF \leftarrow \emptyset$
5: **for each** $block$ **in** $\Psi[b_p, ..., -1]$ **do**
6: $vPoF$.append(validators($block$) $\cap$ validators(tx_{plcl}))
7: $tx_{abort} \leftarrow$ createAbortTx(tx_{plcl}, b_p, $vPoF$)
8: Γ.send(Ω, tx_{abort})
9: Γ.send(Ψ, tx_{abort})
10: **end function**

▷ all $p_i \in P_\Psi$ run abort when receiving any invalid tx_{plcl}
11: **when** tx_{plcl} is received
12: **if** coins_spent(tx_{plcl}) **then** ▷ *some coins in tx_{plcl} were spent, invalid*
13: abort(tx_{plcl})

block of minimum blockheight out of all the blocks that store a transaction spending each of the spent coins, i.e. Proofs-of-Fraud (PoFs). Finally, validators(b/tx) returns the set of validators of block b or transaction tx.

Intuitively, this algorithm proves invalidity by iterating through Ψ, looking for validators that validated both this bulk close and some progress in Ψ that conflicts with it (i.e. some blocks that spent some of the coins). This set of validators is the set of *fraudsters*. Other validators that only validated the transaction might simply have had an old view of the Platypus chain, under the partially synchronous model. Nonetheless, the existence of such block is enough to create the abort transaction, even if the set of fraudsters is empty.

The iteration starts from the block with minimum blockheight of all the Blocks that show that some coin ¢ was transferred from p_i to p_j, for some p_i that claims ownership of ¢, in line 3. The algorithm then continues to account for fraudsters.

V. Correctness & Resilience Optimality

In this Section we prove that Platypus solves the Offchain problem (Section III) and is resilience optimal. By lack of space we defer the full proofs to the companion technical report [26].

Theorem 1 (Correctness)**.** *The Platypus Protocol solves the offchain problem.*

Proof Sketch. The proofs for Algorithm 1 guarantee that m_v validators are requested at all times, all of which explicitly stated to participate as validators, with guaranteed termination if there are enough validators m_v, i.e. the Platypus chain is properly bootstrapped and the security assumptions hold at the end of Algorithm 1. Once Algorithm 1 finishes, the inner consensus of Ψ guarantees the consensus properties, given the assumption $f < m_v/3$, with the same set of validators m_v, and using the same m_0 as threshold for Byzantine behaviour [6]. Finally we show that Algorithm 2, which closes Ψ, solves the offchain problem. The proof of termination is straightforward (see technical report [26]), while those of agreement, ABORT-Validity and COMMIT-Validity are closely related. First, we prove that all COMMIT decisions must be agreed upon by the childchain consensus. Then, we prove that an ABORT can only be proposed (and decided) if a correct process provides a previous transaction that conflicts with the proposed COMMIT. Finally, we show that no two processes decide differently, by contradiction (analogously to the proof of Theorem 2). □

The following theorem shows that our construction works in the strongest coalition the adversary can form.

Theorem 2 (Resilience optimality)**.** *It is impossible to perform a transfer operation with an offchain protocol under partial synchrony if $f \geq m_v/3$.*

Proof. We proceed by contradiction. As a coalition of $f \geq n_v/3$ can corrupt the blockchain, suppose instead that $m_v/3 \leq f < n_v/3$. As there exists a correct offchain protocol Γ that transfers at least one coin ¢ from account a to account b in transaction tx, we look at the amount m_0 of validators needed for the protocol to COMMIT. If the protocol had a threshold of $m_0 > 2m_v/3$ signatures, it follows that some of the processes controlled by the adversary should have agreed to such transaction, which is impossible to enforce.

Thus, m_0 has to be such that $m_0 \leq 2m_v/3$. In such a case, consider a partition of validators V_Ψ into Q_1, Q_2 of correct processes, and F the set of processes controlled by the adversary, such that $Q_1 \cap Q_2 = Q_1 \cap F = F \cap Q_2 = \emptyset$. Let tx' be a transaction transferring the same coin ¢ from account a to c, $c \neq b$ and let $|Q_1| = |Q_2| = \frac{|V_\Psi| - |F|}{2} < m_v/3$. Since $Q_1 \cup Q_2 \cup F = V_\Psi$, we have that $|Q_1| + |Q_2| + |F| = m_v$, meaning that $|Q_1| < m_v/3$ and $|Q_2| < m_v/3$. Therefore, $|F| + |Q_1| > 2m_v/3$ and $|F| + |Q_2| > 2m_v/3$. In this case, if $m_0 < |F| + |Q_2|$, then $m_0 < |F| + |Q_1|$ and thus it would be possible for the adversary to validate tx for Q_1 and tx' for Q_2. Thus, m_0 must be such that $m_0 > |F| + |Q_2| > 2m_v/3$. This is a contradiction: we already showed above that m_0 should be such that $m_0 \leq 2m_v/3$. □

VI. Theoretical Analysis

In this section, we analyze the communication, message and time complexity of the Platypus protocol, ignoring the complexity of the underlying blockchain. We consider the calls to acsend(Ψ, tx) and send($\{\Psi, \Omega\}$, tx) to have the same complexities as one multicast to all validators $V_{\{\Psi,\Omega\}}$ of the blockchain that receives the transaction tx.

1) Message complexity: The message complexity of Algorithms 2, 4 and 5 is $\mathcal{O}(m_v^2)$ and that of Algorithm 3 is $\mathcal{O}(m_p * m_v)$. We conjecture that the complexity could however be reduced to $\mathcal{O}(m_v)$ at some points, leveraging non-interactive aggregation of the validators signatures and messages, but certain calls to acsend(...) and send(...) would

still have a complexity of $\mathcal{O}(m_v^2)$, as they can be executed by all processes. The same applies to Algorithm 1, with the exception that Phase 2 has a message complexity of $\mathcal{O}(m_p * n_p)$, thus being the complexity of Platypus.

2) Communication complexity: The message size is $\mathcal{O}(m_p)$ in lines 18 and 14 of Algorithm 1, leading to a communication complexity of $O(\max\{m_p{*}n_p,\ m_v^3\})$, because of phases 2 and 3 of the algorithm. Line 7 of Algorithm 3 also has a message size of $\mathcal{O}(m_v)$, leading to a communication complexity of $O(m_p * m_v^2)$, although the set of validators can be removed if no punishments are considered. The rest of messages have constant size in all algorithms, thus their communication complexity is the same as their message complexity.

3) Time complexity: The time complexity is $\mathcal{O}(m_v)$ due to Phase 4 of Algorithm 1 and Phase 2 of Algorithm 2. Algorithms 3, 4 and 5 have constant time complexity. Again, we conjecture that, leveraging non-interactive aggregation, the time complexity can be reduced to constant time.

Note that these complexities are lower or comparable to consensus algorithms in the same model [6].

VII. Improvements & Discussion

In this section, we consider additional features of the Platypus chain, and its usage for the general sidechains problem, which we also define.

A. Crosschain payments

A crosschain payment can be of two types, either a payment to a parentchain, or a payment through a parentchain to another childchain. With the above-shown protocol, a payment to a parentchain would require a Platypus bulk close transaction, and a new Platypus creation transaction. We describe an extension of the protocol to perform payments without closing and reopening Platypus chains.

1) Users' Splice-in & Splice-outs: Splice-in and Splice-out transactions allow users to get their funds into and out of the Platypus chain, respectively.

– *Splice in.* Splicing in allows users to join a Platypus chain. Since this transaction takes place after the Platypus chain has been created, it requires some validation by both their sets of validators. Algorithm 4 shows the Splice in protocol for a process p_i that wants to join Ψ. A splice in transaction tx_{spin} must be written in both Ω and Ψ, after which the funds can only be spent in Ψ.

– *Splice out.* The same way users can splice into an existing Platypus chain, they can get their funds back in the parentchain. Again, this is a sensible operation that requires proper synchronization between both Platypus chain and parentchain so as to protect against fraud.

The splice out transaction allows processes to leave a Platypus chain before it is closed, retrieving their funds back in the parentchain. In this case, we require first the transaction to be finalized in Ψ before being considered for the parentchain. Algorithm 5 shows the splice out protocol for a process p_i. This protocol is rather a simplification of Algorithm 2. It creates and tries to write a splice out transaction tx_{spou}, that can be aborted with an abort transaction tx_{abort}.

Algorithm 4 Splice in algorithm for process p_i

▷ State of the algorithm
Ω, Ψ, Γ, the Blockchain, Platypus Blockchain and protocol
$\mathbb{C}_i$, coins that belong to process p_i
$plid$, the Platypus chain identifier
$tx_{spin} \leftarrow \perp$, the splice in transaction

▷ p_i creates and waits for transaction to write
1: $tx_{spin} \leftarrow \mathsf{createSpliceInTx}(\mathbb{C}_i, plid)$
2: $tx_{spin} \leftarrow \mathsf{sign_i}(tx_{spin})$
3: $\Gamma.\mathsf{send}(\Omega, tx_{spin})$
4: $\Gamma.\mathsf{send}(\Psi, tx_{spin})$

Algorithm 5 splice out for process p_i

▷ State of the algorithm
Ω, Ψ, Γ, the Blockchain, Platypus Blockchain and protocol
$\mathbb{C}_i$, the coins that belong to process p_i
$tx_{spou} \leftarrow \perp$, the splice out transaction

▷ p_i creates and waits for transaction to write in Ψ
1: $tx_{spou} \leftarrow \mathsf{createSpliceOutTx}(\mathbb{C}_i)$
2: $tx_{spou} \leftarrow \mathsf{sign_i}(tx_{spou})$
3: $r \leftarrow \Gamma.\mathsf{acsend}(\Psi, tx_{spou})$ ▷ Get back tx_{spou} or tx_{abort}
4: **if** $(r.type = \mathsf{ABORT})$ **then** $\Gamma.\mathsf{send}(\Omega, r.tx_{abort})$
5: **else if** $(r.type = \mathsf{COMMIT})$ **then** $\Gamma.\mathsf{send}(\Omega, r.tx_{spou})$

2) Validators' Splice-in & Splice-outs: Notice that the adversary could gain enough relative power either by splicing in or by correct validators splicing out. One way to prevent this is by keeping the set of validators intact regardless of the funds each validator has after Platypus creation. An additional feature of the protocol might provide explicit delegation of the validator set to other users, similar to how consortium blockchains behave.

Another alternative may allow users and the set of validators to splice in and splice out in a permissionless, Proof-of-Stake based environment. In this set, validators should take great care at identifying the probability of an adversary gaining enough relative power. If the probability of an adversary gaining enough relative power reaches a certain threat threshold, either by the validators set reducing significantly, based on the funds at stake or any other information used for heuristics, validators can generate a Platypus bulk close transaction and safeguard all users' funds. This variation requires the assumption that the adversary never gains enough relative stake such that $stake(f) \geq stake(m_v)/3$.

3) Crosschain payments with splice-in & splice-outs: A crosschain payment in between two blockchains with Platypus is a payment of one user from/into an existing Platypus chain to/from its parentchain, or in between two Platypus chains that share a common parentchain. In section VII-B, we generalize such definition. Regardless of the particular conditions and assumptions for splice-ins and splice-outs, we illustrate in this section how these transactions would work.

∗ *Crosschain payment from/to parentchain*. This case is trivial using Algorithm 4 or 5, respectively.

∗ *Crosschain payment between childchains*. This is performed with a splice out into the common parentchain, fol-

lowed by a splice in into the recipient.

B. Platypus for Sidechains

The childchain definition done in section III can easily be generalized for sidechains by clearly decoupling Ψ from the protocol, and stating different sets for them $P \neq Q$ instead of $P \subsetneq Q$. We define sidechain protocols as a superset of offchain protocols, defined in Section III. A sidechain protocol allows to perform a payment across blockchains, i.e. a *crosschain payment*. If two or more blockchains intend to perform crosschain payments, we refer to them as being sidechains. They may or may not be in a parent-child hierarchy.

$*$ *Sidechain protocol.* Given two blockchains, Ω of P processes and Ψ of Q, $P \neq Q$ a sidechain protocol Π is an offchain protocol that enables transfers in between all accounts p_a, q_a such that $\rho(q_a) = q \in Q, \rho(p_a) = p \in P$. To reflect Ω and Ψ being independent, and this possibility of transferring, we define the following property:

– COMMIT-Matching Knowledge: If a correct process decides COMMIT on a sequence seq of transfer operations in Π between Ω and Ψ, then $\forall p \in P,\ p$ knows a subset seq_1 and $\forall q \in Q,\ q$ knows a subset seq_2, such that seq_1 and seq_2 are two minimal transfer sets, $seq_2 \cap seq_1 = \emptyset$, and it exists one surjective application $f : seq_1 \times seq_2 \rightarrow TR^{-}(seq_1 \cup seq_2) \cup \{0\}$ defined as follows:

$$f(a\,TR\,b, c\,TR\,d) = \begin{Bmatrix} (a\,TR\,d) \text{ if } \rho(b) = \rho(c) = p_i \\ 0 \text{ otherwise} \end{Bmatrix} \quad (2)$$

Also, since the coins are different in different blockchains, we identify coins by their value when calculating the minimal transfer set TR^{-}. Intuitively, for a transaction in seq_1 exists a transaction in seq_2 such that both are transitive (that is, the receiver of one is the sender of the other). If that was not to happen, some of the transactions in seq_1, or in seq_2, would have nothing to do with a payment in between two sidechains.

If $Q \subsetneq P$ then seq_1 is just a set of idempotent transfers of the form $a\,TR\,b$, with $\rho(a) = \rho(b)$, since all $p \in Q$ are also in P, and thus COMMIT-Privacy/Lightness is a particular case scenario of the COMMIT-Matching Knowledge property.

Similarly, if $P \not\supseteq Q$, $P \not\subseteq Q$ then Ω and Ψ are not in the parent-child chain hierarchy.

$*$ *Crosschain payments.* This is solved by our protocol if both sidechains have a common parentchain, as shown in section VII-A3. In general, for a crosschain payment between two unrelated Blockchains Ω_1 and Ω_2, with sets of validators V_{Ω_1} and V_{Ω_2}, they can perform the payment manufacturing an additional Blockchain Φ:

– Create a common parentchain Φ with $V_\Phi \supseteq V_{\Omega_1} \cup V_{\Omega_2}$, extend both their Blockchains to adopt the Platypus protocol, and perform the payment as explained in section VII-A3. In this case, if the adversary tries to double spend the crosschain payment in Ω_2, or in Ω_1, then, as long as $f < |V_\Phi|/3$, the funds will remain in the parentchain Φ.

– Create a common Platypus chain Φ, with $V_\Phi \subseteq V_{\Omega_1} \cap V_{\Omega_1}$, and perform the payment. In such a case, should $f < |V_{\Omega_1}|$ and $f < |V_{\Omega_2}|$, then the adversary could not double spend the funds in Φ and splice out to both Ω_1 and Ω_2.

C. Attacks

Many of the common attacks for synchronous offchain protocols are not applicable in the partially synchronous Platypus [16], [25], [27]. Theorem 2 shows how if the adversary is such that $f \geq m_v/3$ then it can perform a colluding validators attack. We also introduce the *ABA-transfer* attack. If a coin $¢$ was transferred from process p to process q, and later on again to process p, q can try to ABORT any close/splice out in which $¢$ does not belong to him, by using as proof the deprecated transfer $p\,TR\,p$. To cope with this attack, we use session keys in this document, as mentioned in Section III, thus having two different accounts. Another possible solution involves committing to merkle trees and requiring any ABORT to provide a merkle tree T such that the merkle tree T' of the COMMIT attempt is included $T' \subseteq T$ as part of the PoF.

VIII. Related Work

1) Sidechains & childchains: Childchains were introduced with the concept of sidechains [2]. A sidechain targets crosschain payments not necessarily in the parent-child hierarchy. Childchains were first formalized in [12] for an efficient childchain protocol in a semi-synchronous model. Unfortunately, their notion of semi-synchronous communication considers that every messages get delivered in a non-null bounded amount of time Δ, which remains a synchrony assumption [9]. The term 'semi' is used by the authors to denote the fact that the bound Δ is not null. Note that this notion differs from partial synchrony [9] where the bound is unknown.

2) Scalable consensus: Sharding [31], hybrid consensus [11] and consensus on super-blocks [7] also aim at scaling blockchains, with some partially synchronous proposals [20], [23]. However, their constructions hardly consider privacy requirements and crosschain payments.

3) Crosschain payments: Many protocols propose generic crosschain payments. Atomic crosschain swaps [14], [22], [30], [32] typically rely on synchronous Hashed Timelock Contracts [28], while others focus on a crash model, rather than a Byzantine one [30]. Consensus-based crosschain interactions [18], [29] are the closest to our proposal. Some of them fall in the sidechain category. Polkadot [29] reuses the idea to manufacture a common parentchain, to perform payments asynchronously, although it was not proved correct.

Crosschain deals [15] allow for auctions or relaying payments, with both a synchronous and a partially synchronous protocols. Unlike our problem, the crosschain deals problem tolerates that the protocol aborts even if the only processes proposing abort are Byzantine. Our problem disallows such an execution as it could prevent a correct process from cashing out. Our implementation ensures this execution cannot happen by requiring every abort transaction to contain a valid proof of fraud. This makes our offchain problem the first Byzantine fault tolerant variant of the atomic commit problem [5] that has only been defined to our knowledge in a crash model.

4) Offchain protocols: State and payment channels [8], [25] were the first offchain proposals for Blockchains. The Lightning Network [25], a network of channels that relay

payments offchain, is the most notable of the offchain proposals, while other similar offchain payment networks have been proposed [17], [21], some of which work in asynchronous communications [1], [19]. While channel networks scale, they are still limited to the amount of transactions allowed to open and close each of its channels. Lightning Factories allow users to open several channels at once while preserving constant lock-in time [27].

Other works also target more scalability than payment channels through factory-like constructions under different systems and assumptions [4]. All factories and channels require all involved users to explicitly sign to perform transfers, impacting performance. PLASMA is the most known childchain construction, proposed for Ethereum [24]. It provides the first childchain protocol with fraud detection.

These offchain protocols are synchronous, which could make them vulnerable to the Balance Disclosure attack [16] that discloses the balances of other users in the Lightning Network, while the Stale Channel attack locks balances of all users in a channel/factory [27]. There is thus great interest in achieving offchain scalability and privacy in partial synchrony.

IX. Conclusion

The Platypus chain is the first childchain that does not assume synchrony or a trusted execution environment. We discuss its extensions and applications for scalability and for secure crosschain payments. Finally, we showed that our protocol is correct and resilience optimal. As future work, we would like to cope with more than $n/3$ rational processes.

Acknowledgments: This research is supported under Australian Research Council Discovery Projects funding scheme (project number 180104030) and Future Fellowship funding scheme (project number 180100496).

References

[1] G. Avarikioti, E. K. Kogias, and R. Wattenhofer. Brick: Asynchronous state channels. *CoRR*, abs/1905.11360, 2019. http://arxiv.org/abs/1905.11360, last accessed 2019-08-19.

[2] A. Back, M. Corallo, L. Dashjr, M. Friedenbach, G. Maxwell, A. Miller, A. Poelstra, J. Timón, and P. Wuille. Enabling blockchain innovations with pegged sidechains. Technical report, Blockstream, 2014. https://blockstream.com/sidechains.pdf, last accessed 2019-08-19.

[3] D. Boneh, M. Drijvers, and G. Neven. Compact multi-signatures for smaller blockchains. In *Asiacrypt*, pages 435–464, 2018.

[4] C. Burchert, C. Decker, and R. Wattenhofer. Scalable funding of bitcoin micropayment channel networks. *Royal Society open science*, 5(8):180089, 2018.

[5] C. Cachin, R. Guerraoui, and L. Rodrigues. *Introduction to reliable and secure distributed programming*. Springer Science & Business Media, 2011.

[6] T. Crain, V. Gramoli, M. Larrea, and M. Raynal. DBFT: Efficient leaderless byzantine consensus and its application to blockchains. In *NCA'18*, pages 1–8. IEEE, 2018.

[7] T. Crain, C. Natoli, and V. Gramoli. Evaluating the red belly blockchain. *CoRR*, abs/1812.11747, 2018. http://arxiv.org/abs/1812.11747, last accessed 2019-08-19.

[8] C. Decker and R. Wattenhofer. A fast and scalable payment network with bitcoin duplex micropayment channels. In *Symposium on Self-Stabilizing Systems*, pages 3–18. Springer, 2015.

[9] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, 35(2):288–323, 1988.

[10] P. Ekparinya, V. Gramoli, and G. Jourjon. Impact of man-in-the-middle attacks on ethereum. In *37th IEEE Symposium on Reliable Distributed Systems (SRDS)*, pages 11–20, 2018.

[11] I. Eyal, A. E. Gencer, E. G. Sirer, and R. V. Renesse. Bitcoin-NG: A scalable blockchain protocol. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 45–59, Santa Clara, CA, Mar. 2016. USENIX Association.

[12] P. Gazi, A. Kiayias, and D. Zindros. Proof-of-stake sidechains. Technical report, Cryptology ePrint Archive, Report 2018/1239, 2018. https://eprint.iacr.org/2018/1239, last accessed 2019-08-19.

[13] Ö. Gürcan, A. Ranchal-Pedrosa, and S. Tucci-Piergiovanni. On cancellation of transactions in bitcoin-like blockchains. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 516–533. Springer, 2018.

[14] M. Herlihy. Atomic cross-chain swaps. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, pages 245–254. ACM, 2018.

[15] M. Herlihy, B. Liskov, and L. Shrira. Cross-chain deals and adversarial commerce. *CoRR*, abs/1905.09743, 2019. http://arxiv.org/abs/1905.09743, last accessed 2019-08-19.

[16] J. Herrera-Joancomartí, G. Navarro-Arribas, A. Ranchal-Pedrosa, C. Pérez-Solà, and J. Garcia-Alfaro. On the difficulty of hiding the balance of lightning network channels. Technical Report 2019.328, Cryptology ePrint Archive, 2019. https://eprint.iacr.org/2019/328, last accessed 2019-08-19.

[17] R. Khalil, A. Gervais, and G. Felley. NOCUST–a securely scalable commit-chain. Technical Report 642, Cryptology ePrint Archive, 2018. https://eprint.iacr.org/2018/642, last accessed 2019-08-19.

[18] J. Kwon. Tendermint: Consensus without mining. Technical report, relayto, 2014. https://cdn.relayto.com/media/files/LPgoWO18TCeMIggJVakt_tendermint.pdf, last accessed 2019-08-19.

[19] J. Lind, O. Naor, I. Eyal, F. Kelbert, P. Pietzuch, and E. G. Sirer. Teechain: Reducing storage costs on the blockchain with offline payment channels. In *ACM SYSTOR*, pages 125–125, 2018.

[20] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena. A secure sharding protocol for open blockchains. In *ACM SIGSAC CCS*, pages 17–30, 2016.

[21] A. Miller, I. Bentov, R. Kumaresan, and P. McCorry. Sprites: Payment channels that go faster than lightning. *CoRR*, abs/1702.05812, 2017. http://arxiv.org/abs/1905.11360, last accessed 2019-08-19.

[22] T. Nolan. Atomic swaps using cut and choose., 2016. https://bitcointalk.org/index.php?topic=1364951, last accessed 2019-08-19.

[23] R. Pass and E. Shi. Hybrid Consensus: Efficient Consensus in the Permissionless Model. In *31st International Symposium on Distributed Computing (DISC 2017)*, volume 91 of *LIPIcs*, pages 39:1–39:16, 2017.

[24] J. Poon and V. Buterin. Plasma: Scalable autonomous smart contracts. Technical report, Plasma, 2017. https://plasma.io/plasma.pdf, last accessed 2019-08-19.

[25] J. Poon and T. Dryja. The bitcoin lightning network: Scalable off-chain instant payments. Technical report, Bitcoin Lightning, 2016. https://www.bitcoinlightning.com/wp-content/uploads/2018/03/lightning-network-paper.pdf, last accessed 2019-08-19.

[26] A. Ranchal-Pedrosa and V. Gramoli. Platypus: a partially synchronous offchain protocol for blockchains. *CoRR*, abs/1907.03730, 2019. http://arxiv.org/abs/1907.03730, last accessed 2019-08-19.

[27] A. Ranchal-Pedrosa, M. Potop-Butucaru, and S. Tucci-Piergiovanni. Scalable lightning factories for bitcoin. In *ACM/SIGAPP SAC*, pages 302–309, 2019.

[28] R. Russell. Lightning networks part ii: Hashed timelock contracts (htlcs). https://rusty.ozlabs.org/?p=462. https://rusty.ozlabs.org/?p=462, last accessed 2019-08-19.

[29] G. Wood. Polkadot: Vision for a heterogeneous multi-chain framework. *White Paper*, 2016. https://icowhitepapers.co/wp-content/uploads/PolkaDot-Whitepaper.pdf, last accessed 2019-08-19.

[30] V. Zakhary, D. Agrawal, and A. El Abbadi. Atomic commitment across blockchains. *CoRR*, abs/1905.02847, 2019. http://arxiv.org/abs/1905.02847, last accessed 2019-08-19.

[31] M. Zamani, M. Movahedi, and M. Raykova. Rapidchain: Scaling blockchain via full sharding. In *ACM SIGSAC CCS*, pages 931–948, 2018.

[32] A. Zamyatin, D. Harz, J. Lind, P. Panayiotou, A. Gervais, and W. J. Knottenbelt. Xclaim: Trustless, interoperable cryptocurrency-backed assets. Technical Report 2018/643, Cryptology ePrint Archive, 2018. https://eprint.iacr.org/2018/643, last accessed 2019-08-19.