Cross-Chain Communication Using Receipts

Arasu Arun and C. Pandu Rangan

Indian Institute of Technology Madras, Chennai, India
{arasua,rangan}@cse.iitm.ac.in

Abstract. The functioning of blockchain networks can be analyzed and abstracted into simple properties that allow for their usage as blackboxes in cryptographic protocols. One such abstraction is that of the growth of the blockchain over time. In this work, we build on the analysis of Garay et al. [10] to develop an interface of functions that allow us to predict which block a submitted transaction will be added by. For cross-chain applications, we develop similar prediction functions for submitting related transactions to multiple independent networks in parallel. We then define a general "receipt functionality" for blockchains that provides a proof, in the form of a short string, that a particular transaction was added to the blockchain. We use these tools to obtain an efficient solution to the Train-and-Hotel Problem, which asks for a cross-chain booking protocol that allows a user to atomically book a train ticket on one blockchain and a hotel room on another. We formally prove that our protocol satisfies atomicity and liveness. We further highlight the versatility of blockchain receipts by discussing their applicability to general cross-chain communication and multi-party computation. We then detail a construction of "Proof-of-Work receipts" for Proof-of-Work blockchains using efficient and compact zero-knowledge proofs for arithmetic circuits.

Keywords: Cross-Chain Communication · Sharding · Smart Contracts · Zero-Knowledge Proofs · Train-and-Hotel

1 Background

A blockchain is a data structure consisting of transaction strings pooled together into blocks which are chained by hash pointers. Cryptocurrencies, notably Bitcoin launched in 2009, are generally implemented as an append-only, tamper-resistant blockchain, serving as a public ledger, that is operated on by a permissionless peer-to-peer network of users who follow a specified protocol. The protocol specifies the rules of consensus, allowing all users to agree on the state of the ledger at any time. The validity of each transaction, block and hash pointer depends on the protocol and correctness can be verified by any user. Blockchain networks can support smart contracts, which allow for applications beyond simple transfers of funds. The term "smart contract" comes from them being instructions written in code that are enforced by the security of the underlying blockchain network without the involvement of a specific third party. The power of smart contracts range from specifying simple conditions for fund transfers to implementing entire lotteries.

1.1 Analysis of Blockchains

In our protocols, we would like to treat blockchains as a blackbox to which users submit transactions and view the blockchain as an append-only data structure, which grows over time. In order to use blockchains as a primitive in protocols, certain properties and assumptions need to defined. There have been many works formalizing the properties of blockchains [10, 14, 15, 13]. Some of these formalizations deal with the rate of growth of blockchains, how fast transactions can be added onto the blockchain and the underlying assumptions required

to guarantee such properties. In particular, we will follow the notation and definitions of Garay et al. [10], who define properties and prove them with respect to PoW networks. In Section 2, we define basic properties of blockchains networks and develop an interface of functions that we can use in our protocols. This interface allows us to abstract away the implementation specifics of the blockchain and the properties allow us to prove efficiency and correctness guarantees. We emphasize that these properties can be described for general blockchain networks regardless of consensus algorithms and implementation details.

1.2 Cross-Chain Communication

The application ecosystem consists of multiple independent blockchain networks with their own funds and services. Interoperability between such networks, such as the sharing of information and assets, will lead to more applications. The caveat to any such "cross-chain communication" is that there is no natural synchronization mechanism between two independent networks as the users of one network do not have the data of the other. An example of a cross-chain communication application is the atomic swap problem [12], which involves parties who wish to transfer assets on one chain in exchange for assets on another chain. The problem is solved using protocols that ensure the atomicity of these transfers using hashlocks. Another general scenario is when a user wishes to employ the services of one blockchain network in tandem with that of another. An example is the Train-and-Hotel [9] problem detailed below, for which we develop a protocol that ensures atomicity by using "receipts".

Train and Hotel Problem Let there be two independent blockchain networks, one for booking train tickets and one for booking hotel rooms. The train-and-hotel problem [8] asks for a protocol that allows a user to atomically book a train ticket and a hotel room. By atomically, we mean that either both bookings succeed or both fail. A user should not be left with a train ticket but no hotel room, or vice-versa. We will also refer to this problem as the atomic cross-chain booking problem.

Sharding: In most blockchain networks, each node must store and verify every single transaction and this has serious implications to scalability. Sharding is a solution proposed to solving the scalability issues of blockchains [8]. Shards allow the network to be split into smaller blockchains such that a transaction need only be verified by some nodes and not all. A simple example of sharding is splitting the blockchain into assets, such that each type of asset has its own shard. Pertaining to this work, we could have a network that deals with ticket booking and shards are divided by the items to be booked: trains, hotels, theatres and so on. Cross-shard communication is essential for funds and assets in one shard to interact with another. A detailed decription of sharding is provided in [8]. All our solutions can be suitably adapted to work for shards of the same network.

1.3 Receipt Functionality

Transactions and Data We introduce now the notion of certain data being part of a transaction. Cryptocurrencies employ scripting languages, which determine the structure of transactions. For example, the standard payment transaction can be described by (pk_1, pk_2, v) which denotes address pk_1 transferring funds of value v to address pk_2. Transactions in Bitcoin additionally offer an optional metadata parameters which can be any string of the user's choice. Advanced scripting languages, such as that of Ethereum [16] which is Turing Complete, allow transactions to have more complex conditions and instructions. Consider a transaction string txn and any string of data d. The data could be the details of the parties

involved, the transfer value v, the scripts used, the metadata, or any combination of the above. In our protocols, we are concerned with data that can be efficiently verified to belong to a transaction txn by viewing the transaction string alone (that is, without the rest of the blockchain data structure).

Receipts A receipt allows any user to prove that a specific data d is added to a given blockchain $\mathcal{B}$. That is, there is a block in the blockchain $\mathcal{B}$ that contains a transaction txn that contains the data d. For clarity, we will sometimes denote the blockchain data structure as $\langle\mathcal{B}\rangle$. We let $\mathsf{succinct}(\mathcal{B})$ denote some succinct representation of the blockchain. Examples of a succinct representation could be the genesis block or the hash of a known block in the blockchain (discussed further in Section 6).

We represent the receipt functionality as two functions $\Pi_{receipt} = (\mathsf{Generate}, \mathsf{Verify})$. A (non-interactive) proof is generated as $\pi \leftarrow \mathsf{Generate}^{\langle\mathcal{B}\rangle}(\mathsf{succinct}(\mathcal{B}), d)$ where the superscript $\langle\mathcal{B}\rangle$ denotes access to the blockchain data structure. Verification of the proof π does not involve access to the entire blockchain and only requires $\mathsf{succinct}(\mathcal{B})$ as follows: $\{0,1\} \leftarrow \mathsf{Verify}(\mathsf{succinct}(\mathcal{B}), d, \pi)$. We will define the notion of an unforgeable receipt functionality for general blockchains, regardless of the blockchain protocol and its implementation details, in Def 3.1. The goal of this work is to highlight applications of receipts in cross-chain communication. We argue the versatility of receipts to synchronizing two independent networks. We also discuss other applications of receipts in MPC. In Section 6, we provide a construction of receipts for Proof-of-Work blockchains (such as Bitcoin, Ethereum) using zero-knowledge proof techniques for arithmetic circuits

1.4 Zero-Knowledge Proofs

We will give a brief introduction to zero-knowledge proofs-of-knowledge for arithmetic circuits (see [3] for a detailed description). Let there be a circuit C that is known to two parties, Alice and Bob. If Alice possesses a witness w to the circuit such that $C(w) = 1$, a zero-knowledge proof-of-knowledge allows her to produce a non-interactive proof π such that π convinces Bob that Alice possesses a valid witness for C. If Alice does not possess any witness then she can only convince Bob with negligible probability. The zero-knowledge property says that Bob does not learn anything about the witness. We will use the following interface to describe such proofs. Let $\Pi_{PK} = (\mathsf{ProveZK}, \mathsf{VerifyZK})$ be PPT algorithms such that:

- Let C is a circuit and w be an input. If $C(w) = 1$, then $\pi \leftarrow \mathsf{ProveZK}(C, w)$ produces a proof π such that $\mathsf{VerifyZK}(C, \pi) = \texttt{accept}$.
- If Alice does not possess a valid witness for C, then the probability that Alice can generate a proof π such that $\mathsf{VerifyZK}(C, \pi)$ accepts is negligible.

For ease of notation, we omit details like trusted setups and common reference strings. In this work, we are interested in such methods not for their zero-knowledge property, but because of the compact non-interactive proofs they produce. An example of such a proof technique is zk-SNARKs [2]. The technique produces constant-sized proofs independent of the size of the witness (288 bytes). However, one drawback is that it requires a trusted setup. Proof techniques that don't require a trusted setup, but have larger proof sizes, include Bulletproofs [5] and zk-STARKs [1]. We omit a discussion on the verifier and prover complexities, which is available in the respective references. Our protocol is compatible with all these techniques.

1.5 Related Work

There have been many works formalizing the properties of blockchains [10, 14, 15, 13]. The notion of abstracting these properties as functionalities to be used in other protocols have been discussed in [4]. It is known that strong fairness cannot be attained in general multi-party computation under standard cryptographic assumptions [7]. That limit can be overcome by abstracting the blockchain as one trusted third party [4, 6]. In [4], a "claim-or-refund" functionality that allows users to conduct conditional payments is developed and used as a blackbox to provide fairness-with-penalities for general multi-party computation protocols. The idea of proving and verifying publication of transations onto public "bulletin boards" was discussed in [6]. They define the notion of proving that a certain data is published to a signature-based ledger or blockchain and then use witness-encryption to provide strong fairness for general MPCs. A formal description and analysis of the atomic cross-chain swap, along with a solution using hashlocks, is provided in [12]. The train-and-hotel problem was first stated by Andrew Miller [9]. The authors are not aware of any published solution to the problem. We are also not aware of any similar detailed construction of efficient Proof-of-Work blockchain receipts.

1.6 Our Contributions

We build on the properties and analysis of Garay et al. [10] to develop a blacbox-like interface of functions that allow us to provably predict which block number a transaction will be added by. We extend the idea of using receipts on public bulletin boards from [6] to formally define the notion of an unforgeable receipt functionality for general blockchains. We then use this interface and receipt functionality to describe a booking smart contract and an atomic cross-chain booking protocol (the Train-and-Hotel problem) in Section 4. We formally prove the correctness and atomicity of our protocol. To the best of our knowledge, our work is first to provide a formal solution to the Train-and-Hotel problem or develop a similar framework for cross-chain communication. We provide a construction for a receipt functionality in Proof-of-Work blockchains in Section 6 for which we use zero-knowledge proofs to produce compact receipts that are computationally intensive to forge. Our receipts can be attached with transactions and efficiently stored on the blockchain.

2 Abstracting Blockchains

Let $\mathcal{B}$ denote an append-only tamper-resistant blockchain data structure that is operated on by a permissionless peer-to-peer network of users following a specified protocol. Our goal is to formally abstract the properties of a blockchain network and use it as a primitive in our protocols. Doing so allows us to treat the entire network as one blackbox that users submit transactions to and view the blockchain data structure over time. Treating the blockchain as a trusted third party in this manner allows us to derive stronger guarantees for our protocols. Specific abstractions we require are bounds on how many blocks are produced over time and which block a transaction will be added by once submitted. To prove guarantees such as atomicity and liveness for cross-chain communication protocols, we require blocks to not be produced too fast or too slow. Many works have formally analyzed the functioning of blockchains under various protocols and assumptions [10, 14, 15, 13].

2.1 Properties of Blockchains

The notation and definitions we use are from Garay et al. [10]. The properties defined hold for general blockchains. Their analysis is done specific to Proof-of-Work blockchains,

namely Bitcoin, under standard assumptions on network synchronization and honest majority. We emphasize that the following properties can be applied to *all* blockchain networks regardless of the consensus algorithm used and other specific implementation details. Our new derived functions, protocols and proofs are also general and do not make any further assumptions.

Notation

Rounds Rounds are a way to establish a unit of time in a decentralized blockchain network. One round is roughly the time it takes for a message (such as a transaction or a new block) to be propagated throughout the network. Users will submit transactions each round and view the resulting blockchain data structure at the end of each round. The blockchain may grow by 1 block or not grow at all by the end of a round. We will assume that the length of a round for a given blockchain network be a fixed duration t_{round}. Hence, we can talk of a time duration in terms of rounds counts r and the actual length of time $t = r \times t_{round}$. Specific to PoW blockchains, the analysis of [10] assumes that the number of hash queries made by each user in a single round is constant.

Added Transactions and Blocks We say that a block is "added" to the blockchain when it can only get removed from the blockchain (via forking or otherwise) with negligible probability. Specific to PoW networks, a block is considered added only when it is k blocks deep into the chain for some constant k (e.g. 6 in Bitcoin). This term may also be known as "finalized", but as we are abstracting specific details of the consensus protocol, we do not distinguish the two terms.

Security Parameter Let λ be a security parameter. For the blockchain networks we use, the properties below will hold, under appropriate assumptions specific to the blockchain protocol, with probability at least $1 - \nu(\lambda)$ for some negligible function ν. In PoW blockchains, the negligible case arises in unlikely scenarios such as a party finding a collision in a hash function or repeatedly finding multiple Proof-of-Work nonces in consecutive rounds (see [10] for more details).

2.2 Standard Properties

The following three properties are originally defined as the Chain Growth property, Lemma 13 and Liveness property in [10]. We adapt the names and restate them to hide the specifics of their analysis. Let b^* be a constant number dependent on the blockchain. With probability $\geq 1-\nu(\lambda)$, the following three properties will hold for any execution of the blockchain:

Property 1 (Chain Growth - Upper Bound) *There exists a constant r_{upper} such that the blockchain grows by at least b^* blocks after r_{upper} consecutive rounds have passed.*

Property 2 (Chain Growth - Lower Bound) *There exists a constant r_{lower} such that every consecutive stretch of b^* blocks was created over a span of at least r_{lower} consecutive rounds.*

Property 3 (Liveness) *There exists a constant r_{wait} such that a transaction propagated at round R will be added to the blockchain by round $R + r_{wait}$.*

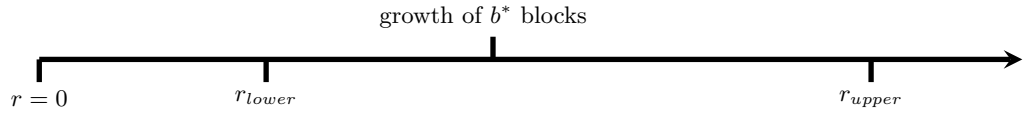

Fig. 1. The blockchain will grow by b^* blocks between r_{lower} and r_{upper} rounds.

Given an arbitrary value of b, we have upper bounds and lower bounds for the number of rounds needed for the blockchain to grow by b blocks as follows:

- The minimum number of rounds needed is: $\lceil \frac{b}{b^*} \rceil r_{lower}$
- The maximum number of rounds needed is: $\lceil \frac{b}{b^*} \rceil r_{upper}$

Given an arbitrary value of r, we have upper bounds and lower bounds for the growth of the blockchain after r consecutive rounds have passed:

- The minimum growth after r rounds is: $\lceil \frac{r}{r_{upper}} \rceil b^*$
- The maximum growth after r rounds is: $\lceil \frac{r}{r_{lower}} \rceil b^*$

Prediction Functions Thus, using the above relations, we define the following prediction functions that we can use in our protocol.

$$b \leftarrow \texttt{max\_growth\_after\_round}(r)$$

- For any r, if $b \leftarrow \texttt{max\_growth\_after\_round}(r)$, then b is the upper bound on the chain growth after any r consecutive rounds.

$$r \leftarrow \texttt{max\_round\_for\_growth}(b)$$

- For any b, if $r \leftarrow \texttt{max\_round\_for\_growth}(b)$, then r is the upper bound on the number of rounds required for the blockchain to grow by b blocks.

We instantiate the functions as follows. The correctness is immediate from the properties defined above.

$$\texttt{max\_growth\_after\_round}(r) = \lceil \frac{r}{r_{lower}} \rceil b^*$$
$$\texttt{max\_round\_for\_growth}(b) = \lceil \frac{b}{b^*} \rceil r_{upper}$$

Note that we can equivalently phrase the above functions using time instead of rounds by converting between round count r and time t using the relation that r consecutive rounds are equivalent to time duration $t = r \times t_{round}$.

2.3 Transaction Addition Parameters

When we submit a transaction to the network, we would like to know which block the transaction will be added by. The Liveness property states that it will be added by r_{wait} rounds, but we do not *exactly* how many blocks will be added by then. However, if we know that the chain will grow by b_{add} only *after* r_{wait} rounds, then b_{add} is our required upper bound. We can obtain b_{add} as follows:

$$b_{add} \leftarrow \texttt{max\_growth\_after\_round}(r_{wait})$$

We can obtain a bound on the the number of rounds r_{add} that b_{add} will be added by as follows:

$$r_{add} \leftarrow \texttt{max\_round\_for\_growth}(b_{add})$$

Thus, knowing the current block height, we have a block number and a round number that our transaction will be added by once submitted. Since these values depend only on the blockchain, we can denote them as follows: $(b^{\mathcal{B}}_{add}, r^{\mathcal{B}}_{add}) \leftarrow \texttt{addition\_parameters}(\mathcal{B})$.

2.4 Analyzing Two Independent Chains

For cross-chain communication, we would like to bound block creation times with respect to two independent blockchains. As the round durations may be different, we state the following properties in terms of **time** as opposed to round counts. Let $\mathcal{B}^1, \mathcal{B}^2$ be two blockchains that satisfy the above properties for parameter values $\{b^{*i}, r^i_{lower}, r^i_{upper}, r^i_{wait}, t^i_{round}\}$ for $i = 1, 2$ respectively. Let $t^i_{wait} = r^i_{wait} \times t^i_{round}$, the maximum waiting time for a transaction to be added on blockchain $\mathcal{B}^i$. We define the following functions:

$$(b_1, b_2) \leftarrow \texttt{max\_growths\_after\_time}(t)$$

- On input numbers b_1, b_2, if $t \leftarrow \texttt{max\_time\_for\_growths}(b_1, b_2)$, then t is the upper bound on time required for $\mathcal{B}^1$ and $\mathcal{B}^2$ to grow by b_1 and b_2 blocks respectively.

$$t \leftarrow \texttt{max\_time\_for\_growths}(b_1, b_2)$$

- On input time duration t, if $(b_1, b_2) \leftarrow \texttt{max\_growths\_after\_time}(t)$, then the upper bound of the growth of blockchains $\mathcal{B}^1$ and $\mathcal{B}^2$ after time t has passed are b_1 and b_2 respectively.

We instantiate the functions as follows. The correctness is immediate from the properties defined above.

$$\texttt{max\_growths\_after\_time}(t) = \max_{i=1,2}\{\texttt{max\_growth\_after\_round}(\lceil \frac{t}{t^i_{round}} \rceil)\}$$
$$\texttt{max\_time\_for\_growths}(b_1, b_2) = \max_{i=1,2}\{\texttt{max\_round\_for\_growth}(b_i) \times t^i_{round}\}$$

2.5 Simultaneous Addition Parameters

In our protocol, we will simultaneously publish transactions $\mathsf{txn}_1, \mathsf{txn}_2$ to blockchains $\mathcal{B}_1, \mathcal{B}_2$ respectively and would like to predict the block numbers they will be added by and what time both blocks will be added by. Just like the single blockchain scenario, we know that the transactions will be added by $t_{wait} = \max\{t^1_{wait}, t^2_{wait}\}$. Again, just like the single blockchain scenario, we cannot predict the exact number of blocks added before t_{wait}. Instead, we can try to predict blocks guaranteed to be added **after** t_{wait} and have these blocks serve as upper bounds for the addition of our transactions. Since these blocks are added after t_{wait}, we then need a new upper bound on the time (denoted as t_{add}) these blocks will be added by.

Let $(b^1_{add}, b^2_{add}) \leftarrow \texttt{max\_growths\_after\_time}(t_{wait})$

- This means that when both submitted transactions are added on their respective chains, the blockchains will grow by length at most b^i_{add} respectively.

  Plugging in the above values b^i_{add}, let $t_{add} \leftarrow \texttt{max\_time\_for\_growths}(b^1_{add}, b^2_{add})$

- This means that both transactions will be added into blocks such that both those blocks will be added to their respective chains by time t_{add}.

Figure 2 illustrates the growth of the blockchains over the specified time durations. Now, when we submit a transaction each to $\mathcal{B}^1, \mathcal{B}^2$ simultaneously, we can predict the block numbers both transactions will be added by and the time duration duration by which both blocks will be added. We will use these functions as an interface to simplify the description and analysis of our protocols. Since these parameters are only dependent on the blockchains, we denote them as follows:

$$(b^1_{add}, b^2_{add}, t^{12}_{add}) \leftarrow \texttt{simultaneous\_addition}(\mathcal{B}^1, \mathcal{B}^2)$$

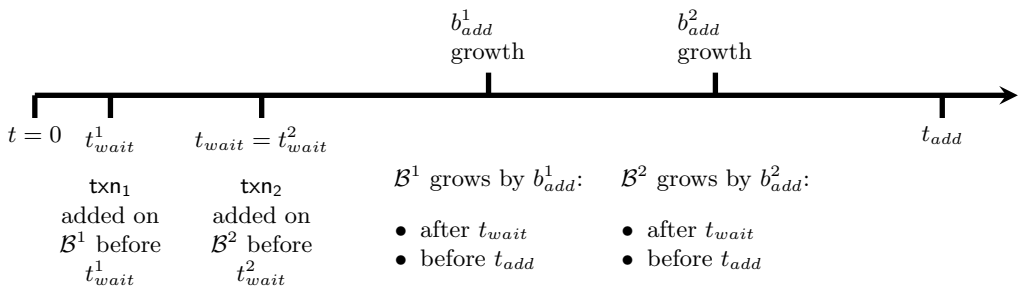

Fig. 2. Simultaneous Addition Parameters (discussed in Section 2.5): A line depicting transaction wait times and block growths over time when submitting two transactions simultaneously to two independent blockchains. The transactions txn_i will be added to the blockchains on or before they grow by b^i_{add} respectively. In this example, we have that $t^1_{wait} < t^2_{wait}$.

3 Receipt Functionality

Let $\mathcal{B}$ be a blockchain data structure that can be succinctly represented using string $\mathsf{succinct}(\mathcal{B})$. For example, $\mathsf{succinct}(\mathcal{B})$ could be the hash of some specific block, such as the genesis block.

Let d be some data such that it can be efficiently decided whether a given transaction txn contains data d or not from just the transaction string (that is, the rest of the blockchain data structure is not required). We will generate a receipt that data d has been added to blockchain $\mathcal{B}$ using a generation function as follows:

$$\pi_d \longleftarrow \mathsf{Generate}^{\langle \mathcal{B} \rangle}(\mathsf{succinct}(\mathcal{B}), d)$$

The superscript denotes access to the blockchain data structure. A boolean verification function works will verify the receipt using just the succinct representation and no access to the blockchain data structure: $\{0,1\} \leftarrow \mathsf{Verify}(\mathsf{succinct}(\mathcal{B}), d, \pi)$.. We formally define the security of such a functionality below.

Definition 3.1 (Unforgeable Receipt Functionality) *Let λ be a security parameter. Let $\mathsf{succinct}(\mathcal{B})$ be a succinct representation of blockchain $\mathcal{B}$. Let $(\mathsf{Generate}, \mathsf{Verify})$ be efficient probabilistic algorithms such that only $\mathsf{Generate}$ has access to the blockchain data structure $\langle \mathcal{B} \rangle$. Let d be some data. $\Pi = (\mathsf{Generate}, \mathsf{Verify})$ is an unforgeable receipt functionality if $\mathsf{Generate}$ produces short proofs π such that:*

- *If there exists a transaction txn with data d in the blockchain $\mathsf{succinct}(\mathcal{B})$, then $\pi \leftarrow \mathsf{Generate}^{\langle \mathcal{B} \rangle}(\mathsf{succinct}(\mathcal{B}), d)$ is such that $\mathsf{Verify}(\mathsf{succinct}(\mathcal{B}), \pi) = 1$ with probability 1. Else, $\perp \leftarrow \mathsf{Generate}^{\langle \mathcal{B} \rangle}(\mathsf{succinct}(\mathcal{B}), d)$.*
- *If $\mathsf{Verify}(\mathsf{succinct}(\mathcal{B}), d, \pi) = 1$, then there exists a transaction txn with data d in the blockchain $\mathsf{succinct}(\mathcal{B})$ with probability $\geq 1 - \nu(\lambda)$.*

Note that the verification function requires access to just the succinct representation and not the entire blockchain. It may be possible for the proof to include additional details, such as the height of the block which the transaction is present in. Section 6 details a construction for Proof-of-Work blockchains.

4 Train-and-Hotel Protocol

We will now describe our protocol for the Train-and-Hotel problem (stated in Section 1.2). We refer to the protocol developed as the Atomic Cross-Chain Booking Protocol.

4.1 Single-Item Booking Protocol

We will first describe the protocol for the booking algorithm for a single item on one blockchain. We will extend this protocol for cross-chain booking in Section 4.2. Let $\mathcal{B}$ be a blockchain where users book items like train tickets or hotel rooms. An item has three states: (1) available for booking, (2) put on hold for a user, (3) booked by a user. Let a *user* wish to book *item* from blockchain $\mathcal{B}$.

Booking Steps A booking involves two steps of one transaction each:

Step 1: (Conditional) Request

Step 2: Confirm or Refund

A user must first "request" an item by paying a deposit. If the request is successful, the item is placed on hold for Δ_{hold} number of blocks. After which, the item is removed from hold and the user must send a fresh request to book it. After a successful hold, the user can then "confirm" the item by paying extra funds, or "refund" the item and receive their initial deposit back. Confirmation and refunds may require particular conditions to be satisfied

(which will be vital for cross-chain booking). The confirmation must be added by Δ_{hold} blocks and refunding can be done anytime. If the user fails to confirm or successfully refund, then the deposit is lost. This is to penalize users who hold items to just delay its booking by other users.

Request Conditions Our protocol uses "conditional requests" that have two conditions:

- $\mathsf{cond}_{\mathsf{confirm}}$ to be satisfied in the confirmation step
- $\mathsf{cond}_{\mathsf{refund}}$ to be satisfied in the refund step

The user can confirm the booking by sending a confirm transaction by Δ_{hold} blocks, satisfying the confirm condition. Else, the user can refund the booking anytime by satisfying the refund condition. Successful requests take some deposit and confirmations cost extra. The refund will return the original deposit back to the user.

Successful and Failed Requests A request can fail if the item is unavailable for booking. When added onto the blockchain, the contract will prepend the request transactions with either "`succ/fail`" depending on whether the hold request was successful. A failed request is **still added** to the blockchain as its receipt can be used as evidence of a failed booking attempt.

Transaction Formats Let id be the unique identifier for this booking (provided by the booking system). The transaction formats are as follows: (we omit metadata and input pointers)

$$\begin{aligned}
\mathsf{txn}_{\mathsf{request}} &= (\texttt{request} \mid id \mid item \mid \mathsf{cond}_{\mathsf{confirm}} \mid \mathsf{cond}_{\mathsf{refund}} \mid \Delta_{\mathsf{hold}}) \\
\mathsf{txn}_{\mathsf{confirm}} &= (\texttt{confirm} \mid \mathsf{key}_{\mathsf{confirm}}) \\
\mathsf{txn}_{\mathsf{refund}} &= (\texttt{refund} \mid \mathsf{key}_{\mathsf{refund}}) \\
\text{(on chain) } \mathsf{txn}_{\mathsf{request}} &= (\texttt{succ/fail} \mid \mathsf{txn}_{\mathsf{request}})
\end{aligned}$$

The confirm and request transactions point to the request transaction. $\mathsf{key}_{\mathsf{x}}$ is the string that is used to satisfy the condition $\mathsf{cond}_{\mathsf{x}}$.

Setting Δ_{hold}: By the Liveness property, once the confirmation transactions are broadcasted, they will be added by r_{wait} rounds. Hence, we must ensure that the blockchain grows by Δ_{hold} blocks only only **after** r_{wait} rounds from when the request is added.

- Set $\Delta_{\mathsf{hold}} = \texttt{max\_growth\_after\_round}(r_{wait})$

Upon seeing a successful request, the user can now confirm the transaction by sending $\mathsf{txn}_{\mathsf{confirm}}$, which is guaranteed to be added in a further r_{wait} rounds by the Liveness property. By the definition of `max_growth_after_round` (discussed in Section 2), the blockchain will grow by Δ_{hold} blocks on or after r_{wait} rounds with overwhelming probability. Hence the user will be able to confirm his booking without being penalized.

4.2 Atomic Cross-Chain Booking Protocol

Let $\mathcal{T}, \mathcal{H}$ be two blockchains that satisfy the Chain Growth and Liveness properties for wait parameter values $\{t^i_{wait}\}$ for $i = \mathcal{T}, \mathcal{H}$ (defined in Section 2.5). We will use superscript i to denote the respective parameters in blockchains $i = \mathcal{T}, \mathcal{H}$. Let $\mathsf{succinct}(\mathcal{T}), \mathsf{succinct}(\mathcal{H})$ be

succinct representations of the blockchains. Let $\Pi = (\mathsf{Generate}, \mathsf{Verify})$ be an Unforgeable Receipt Functionality as per Definition 3.1.

Both blockchains follow the booking protocol detailed in Section 4.1. We now describe the atomic booking protocol:

Protocol Setting Let a *user* wish to book items $item_{\mathcal{T}}$ from $\mathcal{T}$ and $item_{\mathcal{H}}$ from $\mathcal{H}$. Let $id = id_{\mathcal{T}} \parallel id_{\mathcal{H}}$ be the combined unique identifier for the transaction with each part generated by the booking systems of each blockchain. Let $bookid = id_{\mathcal{T}} \parallel id_{\mathcal{H}} \parallel item_{\mathcal{T}} \parallel item_{\mathcal{H}}$.

Atomic Booking Steps The steps of the atomic booking protocol are:

Step 1: Conditionally Request both items

Step 2: Obtain Receipts of successful and failed requests

Step 3: Confirm or Refund bookings

- Confirm if both are successful
- Refund if only one is successful

Below, we will describe the transactions on the $\mathcal{T}$ blockchain. The transactions for $\mathcal{H}$ are analogous.

Step 1. Request Items If a request with the same *id* has already been added in the same blockchain, the booking will be considered invalid. We will discuss an appropriate setting for $\Delta^{\mathsf{i}}_{\mathsf{hold}}$ at the end of the section.

Setting the Conditions We must now set the request and confirm conditions for both bookings. All conditions are such that the user must produce a receipt for specific data. Recall that the booking protocol prepends "`succ/fail`" to a request transaction depending on whether it successful or not. The data and conditions are as follows:

- $\mathsf{d}_{\mathtt{x}} = (\text{“}\mathtt{x}\text{”} \mid \mathtt{request} \mid bookid)$ for $\mathtt{x} = \mathtt{succ,fail}$
- $\mathsf{cond}_{\mathsf{succ}}$: provide a receipt $\pi^{\mathcal{H}}_{\mathsf{succ}}$ of a transaction in $\mathsf{succinct}(\mathcal{H})$ containing $\mathsf{d}_{\mathsf{succ}}$
- $\mathsf{cond}_{\mathsf{fail}}$: provide a receipt $\pi^{\mathcal{H}}_{\mathsf{fail}}$ of a transaction in $\mathsf{succinct}(\mathcal{H})$ containing $\mathsf{d}_{\mathsf{fail}}$

Specifying the Other Blockchain The other blockchain's succinct representation $\mathsf{succinct}(\mathcal{H})$ is specified *by the user* in the conditions. The user can choose any valid succinct representation of any blockchain as long as they can produce valid receipts with respect to it.

The format of *bookid* is $id_{\mathcal{T}} \parallel id_{\mathcal{H}} \parallel item_{\mathcal{T}} \parallel item_{\mathcal{H}}$. Each blockchain will ensure that the receipt from the other blockchain contains the id_i generated by it. This prevents users from using the same failed booking on one chain to refund a booking on the other chain. It also forces the user to produce new receipts and not re-use old ones (discussed further below and in Section 6).

The format of a request transaction is:

$$\mathsf{txn}^{\mathcal{T}}_{\mathsf{request}} = (\mathtt{request} \mid bookid \mid \mathsf{cond}^{\mathcal{T}}_{\mathsf{succ}} \mid \mathsf{cond}^{\mathcal{T}}_{\mathsf{fail}} \mid \Delta^{\mathcal{T}}_{\mathsf{hold}})$$

The train booking can only be confirmed by providing a successful hotel booking. And a train booking is allowed to be refunded by providing evidence of failed hotel request with the same *bookid*.

Step 2. Obtain Receipts Recall that failed request transactions are still added as they serve as evidence of a failed booking. After both requests are added, the receipts can be obtained as follows:

$$\pi_{\mathsf{x}}^{\mathcal{H}} \leftarrow \mathsf{Generate}(\mathsf{succinct}(\mathcal{H}), \mathsf{d}_{\mathsf{x}}) \quad \text{x=}\texttt{succ,fail}$$

Step 3. Confirm or Refund After obtaining the receipts, the user must now confirm or refund his bookings. Three cases arise depending on which of the requests were successful:

- Both unsuccessful: no action is required as no confirmation is possible and no deposit was lost.
- One successful and the other unsuccessful: obtain the receipt of the failed booking from the unsuccessful blockchain and post $\mathsf{txn}_{\mathsf{refund}}$ to refund the successful booking. No action is required on the failed blockchain as no deposit was taken.
- Both successful: confirm both bookings with the corresponding successful receipts.

The transactions are as below (analogous for $\mathcal{H}$):

$$\mathsf{txn}_{\mathsf{confirm}}^{\mathcal{T}} = (\texttt{confirm} \mid \mathsf{key} = \pi_{\mathsf{succ}}^{\mathcal{H}})$$
$$\mathsf{txn}_{\mathsf{refund}}^{\mathcal{T}} = (\texttt{refund} \mid \mathsf{key} = \pi_{\mathsf{fail}}^{\mathcal{H}})$$

Setting Δ_{hold}^{i} The crucial requirement is that once successful requests are added, we must guarantee that confirmations transaction can be added within $\Delta_{\mathsf{hold}}^{\mathsf{i}}$ of the request transaction on each blockchain. Otherwise, the user will be unable to successfully confirm or refund and the deposit will be lost.

$$\text{Set } (\Delta_{hold}^{\mathcal{T}}, \Delta_{hold}^{\mathcal{H}}) \leftarrow \texttt{max\_growths\_after\_time}(2t_{wait})$$

- By the Liveness property, both request transactions will be added by time t_{wait} onto their respective blockchains. If the requests are successful, the user can immediately obtain the receipts and publish confirmations $\mathsf{txn}_{\mathsf{confirm}}^{\mathsf{i}}$.
- By the Liveness property again, these confirmation transactions will be added in further time t_{wait}. Thus, the confirmations will require total time $2t_{wait}$ to be added to the blockchain from the start of the protocol.
- This also immediately implies that they will be added within time $2t_{wait}$ from the addition of the request transactions.
- And by definition of `max_growths_after_time` , the blockchains will each grow by Δ_{hold}^{i} on or after time $2t_{wait}$.

Thus, a user can always successfully confirm an available item. The protocol runtime is $2t_{wait}$. We will formally prove that this setting of parameters provides atomicity and liveness below.

Theorem 4.1 (Atomicity and Liveness) *Let a user proceed to book $item_{\mathcal{T}}$ on blockchain $\mathcal{T}$ and $item_{\mathcal{H}}$ blockchain $\mathcal{H}$ using the Atomic Cross-Chain Booking Protocol from Section 4.2. The user start with funds of value v equal to the total price of the items. We assume that $\mathcal{T}, \mathcal{H}$ are secure blockchains that satisfy the Chain Growth and Liveness properties of Section 2. Let λ be a security parameter and ν a negligible function. With probability $\geq 1 - \nu(\lambda)$, the following Atomicity and Liveness properties hold:*

Atomicity: *One of the following conditions will hold at the end of the protocol:*

- *Both bookings succeed and the user loses funds of value v*
- *Both bookings fail and the user does not lose any funds*

Liveness: *If both items are available, then the user will successfully book both items.*

Proof of Theorem 4.1

Proof. Let $\mathcal{T}, \mathcal{H}$ be two blockchains that satisfy the Chain Growth and Liveness properties for wait parameter values $\{r^i_{wait}, t^i_{round}, t^i_{wait} = r^i_{wait} \times t^i_{round}\}$ for $i = \mathcal{T}, \mathcal{H}$. Let $t_{wait} = \max\{t^{\mathcal{T}}_{wait}, t^{\mathcal{H}}_{wait}\}$. Superscripts i, j denotes blockchains $i, j = \mathcal{T}, \mathcal{H}$. Let $\Pi = (\mathsf{Generate}, \mathsf{Verify})$ be an Unforgeable Receipt Functionality as per Definition 3.1.

Let the protocol start at time 0. We have three cases depending on which of the $\mathcal{T}, \mathcal{H}$ requests are successful:

1. Both are unsuccessful.
2. One is successful, the other is unsuccessful.
3. Both are successful.

Case 1: Both Unsuccessful. In this case, the user need not do anything further as no funds are lost. Atomicity holds.

Case 2: Only One Successful. Without loss of generality, let the successful request be on chain $\mathcal{T}$. Since $\mathcal{H}$ is unsuccessful, no deposit is taken from that blockchain and no further action is required there. However, the user needs to be able to refund the booking on $\mathcal{T}$ to recover the deposit. This is done by obtaining the failure receipt $\pi^{\mathcal{H}}_{\mathsf{fail}}$ from $\mathcal{H}$ and posting a refund transaction on $\mathcal{T}$. Since this is a valid refund, it will eventually be added to the blockchain and we are ensured that the user ultimately loses no funds. Since the hotel booking is unsuccessful, there only way for the user to satisfy $\mathsf{cond}^{\mathcal{T}}_{\mathsf{confirm}}$ and confirm the train booking is by forging a receipt for $\mathcal{H}, \mathsf{d}_{\mathsf{succ}}$ which can only happen with negligible probability. Hence, atomicity holds with overwhelming probability as the user loses no funds and both bookings become unsuccessful.

Case 3: Both Successful. The request transactions will be added by t_{wait} with overwhelming probability. Since both are successful, the user obtains the two receipts $\pi^{\mathsf{i}}_{\mathsf{succ}}$ which act as keys for the other blockchain's confirmation condition $\mathsf{cond}^{\mathsf{j}}_{\mathsf{confirm}}$, validating transactions $\mathsf{txn}^{\mathsf{j}}_{\mathsf{confirm}}$. Since a valid transaction will always be added to the blockchain, atomicity holds. What remains to be proved is that the user must always be able to have both confirmations $\mathsf{txn}^{\mathsf{i}}_{\mathsf{confirm}}$ added within $\Delta^{\mathsf{i}}_{\mathsf{hold}}$ blocks of the addition of the request transactions. Because if that does not happen, then the user will be unable to successfully confirm or refund and lose his deposit forever. We will now show that this will happen with overwhelming probability by the blockchain properties: (the following events will all happen with overwhelming probability)

Setting of Δ_{hold}: The key requirement is that the value of Δ^i_{hold} is far enough to ensure that, within Δ^i_{hold} blocks of a successful request on the first blockchain, the following must all happen: (1) the request will be added on the second blockchain, (2) the receipt can be obtained and (3) the confirmation transaction will be broadcasted and added on the first blockchain.

- We know that all valid transactions, including successful and failed requests, will be added by t^i_{wait} for each blockchain. Thus, t_{wait} is an upper bound on the times both requests will be added by.
- If the requests are successful, the user will then broadcast $\mathsf{txn}^{\mathsf{i}}_{\mathsf{confirm}}$ transactions immediately. These will take a futher t_{wait} time to be added.
- Thus, the total time taken for the request and confirmation is $2t_{wait}$.

$$\text{Let } (\Delta^{\mathcal{T}}_{hold}, \Delta^{\mathcal{H}}_{hold}) \leftarrow \texttt{max\_growths\_after\_time}(2t_{wait})$$

- By the property of `max_growths_after_time`, the blockchains will grow by atmost Δ^i_{hold} each in time $2t_{wait}$.
- Since the confirmations will be added by time $2t_{wait}$ of the broadcast of the request transactions, both confirmations will definitely be added on or within $(\Delta^{\mathcal{T}}_{\mathsf{hold}}, \Delta^{\mathcal{H}}_{\mathsf{hold}})$ blocks from the broadcast of the request transactions.
- This immediately implies they will be added on or within $(\Delta^{\mathcal{T}}_{\mathsf{hold}}, \Delta^{\mathcal{H}}_{\mathsf{hold}})$ blocks from the *addition* of the request transactions.

Thus, under that setting of Δ^i_{hold}, an honest user will always be able to confirm when both requests are successful with overwhelming probability, guaranteeing both liveness and atomicity.

We also have that the total runtime of the protocol is $2t_{wait}$.

5 Other Applications

We informally detail more applications of receipts, highlighting their versatile use in blockchains protocols and multi-party computation.

5.1 Generalization for Cross-Chain Communication

We used receipts to solve the Train-and-Hotel problem. Although the receipts were for booking requests, the structure of the receipts and the protocol itself isn't specific to just booking items. What we really required was to separate all executions of the protocol into disjoint conditions and be able to non-interactively verify a condition of one chain on another. Doing so, we were able to synchronize the two chains and ensure atomicity. That is, following two components are the key reasons to why receipts were effective:

- We split the protocol into different conditions each with some lockable state.
- Each condition can be verified non-interactively using receipts.

Here, the conditions are which of the parallel booking requests end up successful. And the verification is the checking of the respective receipt from the other chain. The locking of state is that of an item being available, booked, or put on hold for a duration. In this way, we believe that receipts can facilitate cross-chain communication for more general applications.

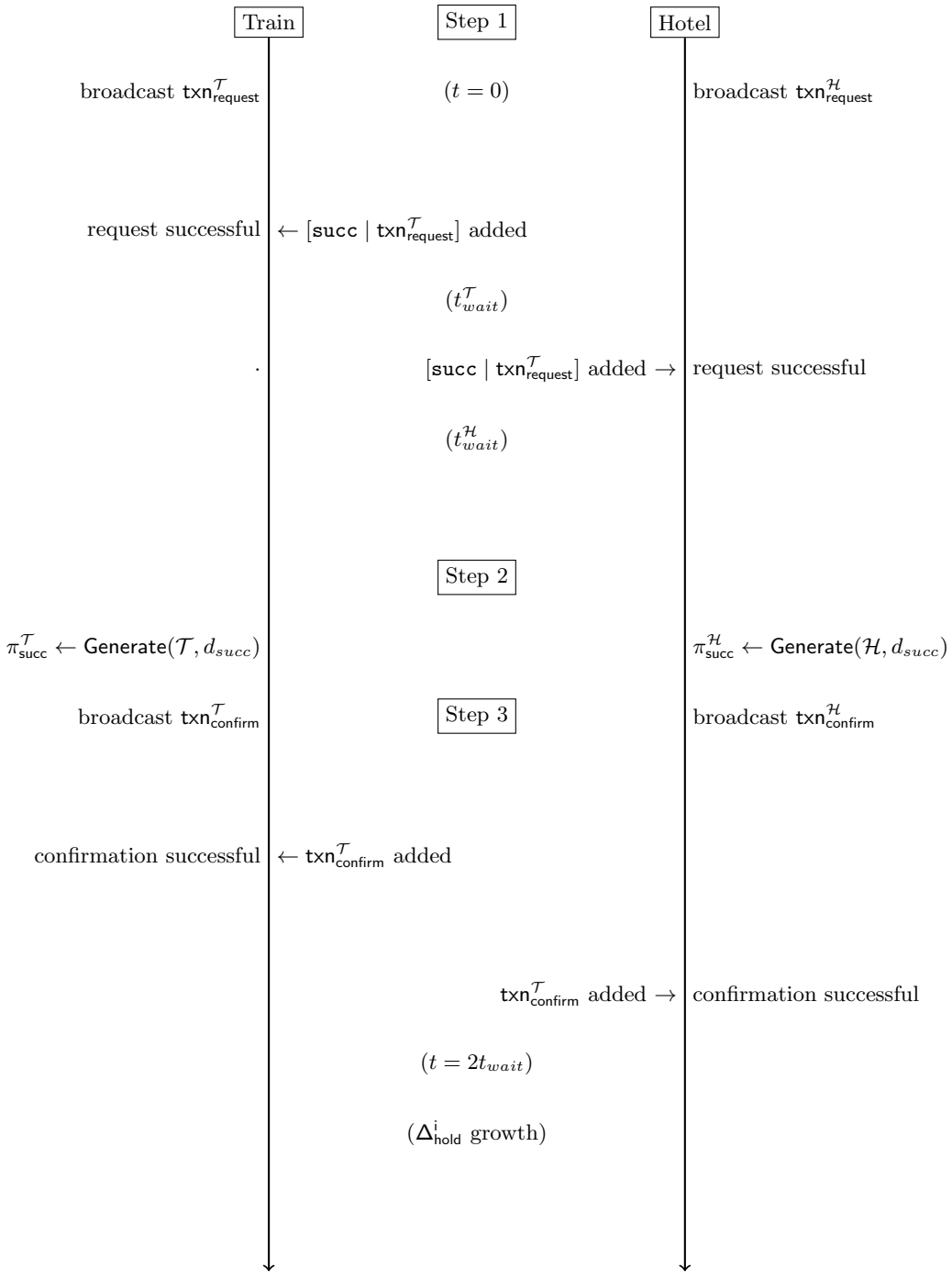

Fig. 3. (**Atomic Cross-Chain Booking Protocol**) The above depicts a possible execution in the case where both requests are successful. The vertical lines denote the blockchains with time increasing downwards. The outer message are actions of the user. The inner messages indicate transactions added onto the blockchain. (t) indicate the time passed.

5.2 Strong Fairness Using Receipts

A multi-party computation (MPC) protocol is said to have strong fairness if either all users obtain the output or no user obtains the output. It cannot be the case that some user (malicious or otherwise) learns the output but another user does not. It is known to be impossible to obtain strong fairness without a trusted third party using standard cryptographic assumptions [7]. As discussed in Section 2, a blockchain network can be treated as one whole trusted third party, allowing us to overcome this barrier. We now use blockchains, and receipts in particular, as a primitive to provide fairness for general MPCs.

A receipt can only be generated by adding a transaction to the blockchain. Since the blockchain is public, the production of a receipt forces parties to reveal certain information to the public. Choudhuri et al. [6] use this idea to force participants of an MPC to produce proofs of publishing data to a public bulletin board. Their idea can be rephrased in our notion of receipts for general blockchains. An informal overview of their algorithm is below: (we refer to [6] for a detailed description with proofs). Let there be n parties who wish to compute a function value of their private inputs.

1. The n participants of the protocol agree on a blockchain $\mathcal{B}$ and succinct representation $\mathsf{succinct}(\mathcal{B})$ at the start of the protocol.
2. At the end of the protocol, each party i receives private values y_i. A witness-encryption (see [6] or [11] for a formal definition) of the protocol's output is also provided to each user.
3. The witness-encrypted output can only be decrypted using a receipt $\pi_{\mathcal{B},y}$ of a transaction that posts all n private values onto the blockchain $\mathcal{B}$. That is, the receipt must of be of data $y = (y_1 \mid \ldots \mid y_n)$ where the y_i's form a valid set of outputs for the protocol.
4. All users share their private values y_i with each other. Even if a malicious party does not share their value, they must eventually post onto the blockchain to obtain the receipt. Else, it is impossible to decrypt the output.
5. Once a user posts the correct transaction onto the (public) blockchain, all users can obtain the receipt and decrypt the computation's output, ensuring strong fairness.

6 Construction of Proof-of-Work Receipts

We now provide a construction of the receipt functionality for Proof-of-Work blockchains. We require that forging a receipt be as computationally intensive as the PoW mining process. We also require the receipts to be compact so they can be stored in transactions efficiently.

Succinct Representation We first explain how a succinct representation of a blockchain $\mathcal{B}$ is created. In our construction, the succinct representation will include the block hash of a given block in the blockchain. The hash of this "base block" B_0, $\mathsf{hash}(B_0)$ will serve as the succinct representation. For efficienct proof generation, this block should be some recently confirmed block. In PoW consensus, each block hash must start with a prefix of 0s of some long length l. We will also add this value l to the succinct representation: $\mathsf{succinct}(\mathcal{B}) = (\mathsf{hash}(B_0), l)$. In cross-chain communication protocols, the user will choose some block B_0 and commit to succinct representation and must provide all receipts with respect to it.

Proof Content Let the data d be stored in transaction txn which is present in a block B_h of height h from the block B_0 used in the succinct representation. In B_h, the hash of txn is stored in the leaf of a Merkle tree of root r. Let p be the set of nodes in the Merkle tree required to verify the presence of leaf node txn in the tree - that is, its ancestors and their

siblings. The root r is present in the block header $\mathsf{header}(B_h)$ whose hash is the block hash $\mathsf{hash}(B_h)$. Thus, we have the following information:

1. The data d is stored in transaction txn
2. p is the set of nodes in the Merkle tree required to verify the presence of leaf node txn
3. r is the root of the Merkle tree, stored in the header $\mathsf{header}(B_h)$
4. The root, the previous block hash, nonce along with other parameters are hashed together to form the block hash $\mathsf{hash}(B_h)$.
5. Let the previous block hash be $\mathsf{hash}(B_{h-1})$ and that block's header be $\mathsf{header}(B_{h-1})$.
6. The header of B_{h-1} now points to previous block hash $\mathsf{hash}(B_{h-2})$ whose header is $\mathsf{header}(B_{h-2})$.
7. This chain continues until block B_1's header points to the succinct representation $\mathsf{hash}(B_0)$.

We can now construct the following string [1]:

$$w = \mathsf{txn} \mid p \mid r \mid \mathsf{header}(B_h), \mathsf{hash}(B_h) \mid \mathsf{header}(B_{h-1}), \mathsf{hash}(B_{h-1}) \mid \ldots \mid \mathsf{hash}(B_0)$$

This string serves as a witness to the fact that d is present in the blockchain. With h as a parameter, the verification algorithm will check the following:

$\mathsf{VerifyWitness_h}(\mathsf{succinct}(\mathcal{B}), d, w)$:

1. Parse w as described above.
2. Verify that txn contains data d.
3. Verify that all hash values, hash pointers are consistent.
4. Verify that the block hashes have a prefix of 0s of length l.

Although this could be a valid receipt, the size of w is very long and infeasible to attach with transactions on the blockchain.

Compressing the Proof Since the verification is done by testing multiple hashes, we can represent the verification algorithm as an arithmetic circuit as follows:

- $C_h(\mathsf{succinct}(\mathcal{B}), d, w)$ is an arithmetic equivalent to $\mathsf{VerifyWitness}_h$ above.
- Note that the structure and input size (w in particular) of C_h varies only with h

Now, fixing inputs $\mathsf{succinct}(\mathcal{B})$ and d, we get a new circuit $C_{h,\mathsf{succinct}(\mathcal{B}),d}(w)$ that only takes witness strings w as inputs. If w is valid proof constructed by the above method, then $C_{h,\mathsf{succinct}(\mathcal{B}),d}(w) = 1$. This now lets us turn to zero-knowledge proof techniques that non-interactively prove knowledge of a valid witness (described in Section 1.4).

Let $\Pi_{zk} = (\mathsf{ProveZK}, \mathsf{VerifyZK})$ be a secure non-interactive zero-knowledge proof-of-knowledge protocol. The functions for a Proof-of-Work Receipt are now as follows:

$\Pi_{receipt} = (\mathsf{GenCircuit}, \mathsf{Generate}, \mathsf{Verify})$:

$\mathbf{GenCircuit}(h, \mathsf{succinct}(\mathcal{B}), d)$:

1. Generate arithmetic circuit C_h equivalent to $\mathsf{VerifyWitness}_h$

[1] In PoW, it must also be proved that B_h is final by showing that it is k blocks deep into the blockchain. This just requires k additional header-hash pairs.

2. Fix the first two inputs of C_h as $\mathsf{succinct}(\mathcal{B}), d$ to obtain circuit $C_{h,\mathsf{succinct}(\mathcal{B}),d}$
3. Output $C_{h,\mathsf{succinct}(\mathcal{B}),d}$

Generate$(\mathcal{B}, \mathsf{succinct}(\mathcal{B}), d)$:

1. Obtain the witness string w as described above.
2. Let $C \leftarrow \mathsf{GenCircuit}(h, \mathsf{succinct}(\mathcal{B}), d)$
3. Output $(\pi \leftarrow \mathsf{ProveZK}(C, w), h)$

Verify$(\mathsf{succinct}(\mathcal{B}), d, (\pi, h))$:

1. Let $C \leftarrow \mathsf{GenCircuit}(h, \mathsf{succinct}(\mathcal{B}), d)$
2. Output $\mathsf{VerifyNIZK}(C, \pi)$

Unforgeability A receipt can be forged by either cheating the zero-knowledge proof system (which can happen with only negligble probability) or creating a witness w without actually posting to the blockchain. Indeed, it is possible for the user to forge a receipt by producing a string w of valid hash values that satisfy the verification algorithm above. However, this is as hard as finding a sequence of Proof-of-Work nonces and hash pointers of long 0-prefix that start from some hash $\mathsf{hash}(B_0)$ and include a transaction txn containing data d. The difficulty is further increased when h is required to be large (say, at least 6). Thus, forging a receipt for a Proof-of-Work blockchain is a computationally intensive task. In our protocols, we will assume that the users cannot perform such a task, especially under the time constraints imposed by the protocols.

Ensuring Recent Receipts Various additional protocol-specific constraints can be introduced to ensure that receipts must be recent and that users cannot use pre-prepared receipts. In the Atomic Cross-Chain Booking protocol of Section 4.2, the value of *bookid* will, among other parameters, include a value $id_{\mathcal{T}} \parallel id_{\mathcal{H}}$ that is uniquely and newly generated by each blockchain's booking system. Since the generation of a valid receipt can only be done after that point, this limits the time to perform the work required to forge a Proof-of-Work receipt.

7 Conclusion

In this work, we develop a framework of prediction functions that allow us to develop cross-chain communication protocol with provable guarantees such as atomicity and liveness. We also defined a receipt functionality for general blockchain networks. A receipt allows a user to efficiently and non-interactively prove that a certain transaction was added on a blockchain. The key application of our receipt functionality is to provide a protocol for the Train-and-Hotel problem, which allows a user to atomically book items across two independent chains. To highlight the applicability of receipts, we discussed how receipts can be used to synchronize two independent blockchain and to achieve strong fairness in MPCs. Finally, we provide a construction of Proof-of-Work receipts for Proof-of-Work blockchains that are short and efficient, using zk-SNARKs. The compactness of our receipts allows them to be efficiently stored in transactions on blockchains.

References

1. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable zero knowledge with no trusted setup. In: Boldyreva, A., Micciancio, D. (eds.) Advances in Cryptology – CRYPTO 2019. pp. 701–732. Springer International Publishing, Cham (2019)
2. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E.C., Virza, M.: Snarks for c: Verifying program executions succinctly and in zero knowledge. In: IACR Cryptology ePrint Archive (2013)
3. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Succinct non-interactive zero knowledge for a von neumann architecture. In: 23rd USENIX Security Symposium (USENIX Security 14). pp. 781–796. USENIX Association, San Diego, CA (Aug 2014), https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/ben-sasson
4. Bentov, I., Kumaresan, R.: How to use bitcoin to design fair protocols. In: Garay, J.A., Gennaro, R. (eds.) Advances in Cryptology – CRYPTO 2014. pp. 421–439. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)
5. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. 2018 IEEE Symposium on Security and Privacy (SP) pp. 315–334 (2017)
6. Choudhuri, A.R., Green, M., Jain, A., Kaptchuk, G., Miers, I.: Fairness in an unfair world: Fair multiparty computation from public bulletin boards. In: ACM Conference on Computer and Communications Security (2017)
7. Cleve, R.: Limits on the security of coin flips when half the processors are faulty (extended abstract). In: STOC (1986)
8. Ethereum: Ethereum sharding faq. https://github.com/ethereum/wiki/wiki/Sharding-FAQ (2019), [Online; accessed 21-September-2019]
9. Ethereum: What is the train-and-hotel problem? https://github.com/ethereum/wiki/wiki/Sharding-FAQ#what-is-the-train-and-hotel-problem (2019), [Online; accessed 21-September-2019]
10. Garay, J., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: Analysis and applications. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology - EUROCRYPT 2015. pp. 281–310. Springer Berlin Heidelberg, Berlin, Heidelberg (2015)
11. Garg, S., Gentry, C., Sahai, A., Waters, B.: Witness encryption and its applications. In: IACR Cryptology ePrint Archive (2013)
12. Herlihy, M.: Atomic cross-chain swaps. In: PODC (2018)
13. Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: A provably secure proof-of-stake blockchain protocol. Cryptology ePrint Archive, Report 2016/889 (2016), https://eprint.iacr.org/2016/889
14. Pass, R., Seeman, L., Shelat, A.: Analysis of the blockchain protocol in asynchronous networks. In: Coron, J.S., Nielsen, J.B. (eds.) Advances in Cryptology – EUROCRYPT 2017. pp. 643–673. Springer International Publishing, Cham (2017)
15. Pass, R., Shi, E.: The sleepy model of consensus. In: Takagi, T., Peyrin, T. (eds.) Advances in Cryptology – ASIACRYPT 2017. pp. 380–409. Springer International Publishing, Cham (2017)
16. Wood, D.D.: Ethereum: A secure decentralised generalised transaction ledger (2014)