Report on ‘Bitcoin as a Transaction Ledger: A Composable Treatment’

Mart Simisker

1. June 2018

Abstract

This report gives an overview of the work done by Badertscher *et al.*, as they put forth a simulation based proof of security of Bitcoin in the universally composable model of Canetti *et al.* For this they specify the goal of the Bitcoin as a ledger functionality in universally composable model. They prove that their ledger functionality is securely UC realised under standard assumptions by an appropriate abstraction of the Bitcoin as a UC protocol.

1 Introduction

This report will give an overview of the article (from hereon ‘the article’) ‘Bitcoin as a Transaction Ledger: A Composable Treatment’ by C. Badertscher, U. Maurer, D. Tschudi and V. Zikas[1]. This report is based on the extended version of this article.

The article follows from the ideas of using Bitcoin as a service for cryptographic protocols. The need to formally define and prove the security of such protocols requires clear definition of the scheme. However, some of the previous, property-based proofs are too strong, as they allow the adversary too little interference with the model. The article promises to specify a new transaction ledger functionality, which guarantees the properties from ([2] and [3]), but also proves, that an abstraction of the Bitcoin protocol also implements this protocol. It is mentioned, that this is the first universally composable (simulation-based) security proof of Bitcoin in the model of Canetti *et al* [4].

The introduction of this new transaction ledger is divided into three sections. In the original article, section 2 presents the model for the real world. In section 3, the ledger functionality in the ideal world is given. Finally, in section 4, a hybrid-world, the world which connects the real world with the ideal world is presented, the main theorem of the paper is given and an intuition of the proof is given. In section 5, real world implementations are discussed. This report will try to follow the same order to avoid unneeded confusion.

1.1 Universally Composable Security Framework

The paper uses UC security framework for representing and analysing security of cryptographic protocols described by Canetti in [5].

In this framework, there are protocols and a method for composing these protocols. The composition operation is called universal composition and definitions of security in this framework are called universally composable (UC). The model of protocol execution consists of a set of interacting computing elements, formally modelled as interactive Turing machines. We say a protocol π securely realizes a function f if for any adversary A, there exists a simulator S such that the adversary A and the protocol π is indistinguishable from the simulator S and the ideal process of f for any environment that outputs a bit indicating it interacting with f or its ideal process. It is said that a protocol UC-realises a function if the protocol securely realises the function with respect to this type of interactive environment.

Let π be some arbitrary protocol, where the parties make ideal calls to some ideal functionality $\mathcal{F}$, or even multiple instances of $\mathcal{F}$, which are running at the same time and are called $\mathcal{F}$-hybrid protocols. Let ρ be a protocol, that UC-realises $\mathcal{F}$. Let π^ρ be a composed protocol π^ρ, where invocations of $\mathcal{F}$ have been replaced with invocations of a new instance of ρ and inputs given to $\mathcal{F}$ are now given to ρ. Informally, the universal composition theorem states that, running this composed protocol π^ρ, with no access to $\mathcal{F}$, has essentially the same effect as running the original $\mathcal{F}$-hybrid protocol π. It guarantees, that for any adversary $\mathcal{A}$, there exists an adversary $\mathcal{A}_\mathcal{F}$ such that no environment machine can tell whether it is interacting with $\mathcal{A}$ and parties running π^ρ or with $\mathcal{A}_\mathcal{F}$ and parties running π. If π UC-realizes some ideal functionality $\mathcal{G}$ then so does π^ρ.

2 Notation

We use the following notation:

A - central adversary

P - set of all players

p_i - used to refer to a given players id.

$||$ - in the context of vectors, is used to denote adding a new element (example: $\vec{M}||m_{new}$)

$[n]$ - set $\{1, ..., n\}$

3 The Model

This section describes the new UC-based model of execution, for the Bitcoin protocol. In the article, Bitcoin miners are represented as players in a multi-party computation. The parties communicate by sending messages over a unauthenticated multicast network with eventual delivery. They can also communicate with a random oracle. It is mentioned, that in UC, available resources are described as hybrid functionalities.

Functionalities with dynamic party sets As opposed to fixed sets of parties, in order to model the dynamic nature of the Bitcoin, miners must be able to join and leave at will. This also means, that the adversary must be notified of currently active miners to be able to corrupt them. For this reason, there are *register* and *de-register* functionality which allow a player to join or leave. The functionalities *is-registered* and *get-registered* are provided to check, whether a player is registered to the functionality. The other is required for the adversary to get the set of registered players. To allow functionalities to connect to shared functionalities, *register*, *de-register* and *get-registered-F* instructions are provided for functionalities.

Communication network In the paper, the miners are assumed to exist in network such, that every miner chooses a sub-set of miners, who are its neighbours. These neighbours are used by the miner to send messages to all other miners. The messages get passed on, and at every step a 'relay link' checks, that such a message has not been received yet. The following assumptions about communication channels are made:

- Guaranteed delivery of messages within a delay parameter Δ but otherwise specified as asynchronous (so that no protocol could rely on timings regarding the delivery of messages). The adversary can delay the messages (by at most Δ) and can use this to change the order, in which messages are sent to some party.
- Only the message is transported. (No information about the sender)
- No privacy guarantees from the channel. The adversary can read all messages on the channel.

The communication is assumed as every miner $p_j \in P$ has access to F_{U-CH}, a multi-use unicast channel, with receiver p_j, such that any miner p_i can input messages to that channel. The messages get unique id-s as they enter the channel. The adversary can then see the message content and delay any message by a finite amount (up to Δ). To avoid the adversary delaying only messages sent by the honest parties, the channel is turned into a fetch message mode. This means, that the channel delivers a message only if it receives a special fetch command. The delaying is then introduced with a value T_{mid} (the delay for message mid), for which the channel ignores the next T_{mid} fetch commands for this message. Also the accumulative delay on any message on a channel can not be higher than Δ. The adversary is allowed to accumulate these delays in parts and can also send negative delays to submit the message in the next fetch. The adversary can reorder the messages by sending a swap command.

It is mentioned, that the Bitcoin uses this unicast network to achieve multicast mechanism. The main differences from the unicast model are, that a message is recorded $|P|$ times (once for each possible receiver). The adversary can delay any subset of the messages (but still regarding Δ). The adversary can also send messages to chosen subsets of the parties.

Random oracle To model queries to the hash function, the random oracle functionality F_{RO} is introduced. Given a value x, if x has not been queried before, it will be chosen uniformly at random, mapped internally and sent to the party. In case it has been queried before, it will be read from the mapping.

The clock functionality To cast synchronous protocols in UC, Katz *et al.*[6] have proposed a methodology called the clock. The clock protocol ensures that all parties will proceed in synchronous rounds. The commands, 'clock read' and 'clock update' are mentioned as interfaces to the clock functionality. It is important, that all the parties have access to the clock. The clock works in rounds and will not proceed to the next round until all the parties have submitted 'clock update' commands. The definition 3 should capture in generic matter the predictable behaviour of pattern updates a protocol must do before sending the clock update command. The definition uses definitions of, honest input sequence $\vec{\mathcal{I}}_H = ((x_1, pid_1), ..., (x_m, pid_m))$ given to the honest parties during a step. Timed honest-input sequence, $\vec{\mathcal{I}}_H^T$, extends the pairs in $\vec{\mathcal{I}}_H$ by adding τ_i, the time of the global clock at the time when the input was given to receiver. (Note that x_i is the i-th input, pid_i, is the identity of the receiving party).

Definition : A $\mathbf{G}_{\text{CLOCK}}$-hybrid protocol Π has a predictable synchronization pattern *iff* there exist an *algorithm* predict-time$_\Pi(\cdot)$ such that for any possible execution of Π, the following holds: If $\vec{\mathcal{I}}_H^T = ((x_1, pid_1, \tau_1), ..., (x_m, pid_m, \tau_m))$ is the corresponding timed honest-input sequence for this execution, then for any $i \in [m-1]$:

$$\text{predict-time}_\Pi((x_1, pid_1, \tau_1), ..., (x_i, pid_i, \tau_i)) = \tau_{i+1}$$

Assumptions Assumptions are often used in security proofs. However, in case of simulation based proof, restricting the class of adversaries and environments would mean that the UC composition theorem can no longer be generally applied. This would dismiss the idea of using a simulation-based security proof. In this case, a solution, which uses an additional parameter q is introduced. q will be used to bound the number of activations a corrupted party can make to the Random Oracle during a round.

4 Transaction-Ledger functionality

This section will describe the proposed ledger functionality ($\mathcal{G}_{\text{LEDGER}}$).

4.1 The building blocks

Anyone can submit a transaction. The transaction is then validated by means of a predicate **Validate** and if valid, added to a buffer *buffer*. The adversary $\mathcal{A}$ is informed upon receiving a transaction and given the contents. The functionality $\mathcal{G}_{\text{LEDGER}}$ periodically creates a block by taking transactions from the *buffer* and applying **Blockify**. The block is then added to its permanent state *state*. The *state* is described as a structure, which contains a part of the blockchain, which the adversary can no longer change (corresponds to common prefix). All parties can request to read this *state*.

In the following part, different relaxations to the ledger are discussed. In [7] it is stated, that transactions in the buffer can not conflict with one another. As this is impossible to provide, due to network delay, the authos of this paper allow conflicting transactions in the buffer during **Validate**, however the transactions in the *buffer* must not conflict with the *state*. Upon creating a block, the *buffer* is revalidated and conflicting transactions are removed.

Another change, is that the state does not have to be updated at a fixed period, but the adversary is allowed to define, when the update occurs. This is done, by allowing the adversary to propose new blocks **NxtBc**. To avoid giving the adversary too much power and to keep liveness and chain quality properties, a new algorithm, **ExtendPolicy** is introduced. If the adversary does not comply with the policy, the **ExtendPolicy** will propose the new block instead. **ExtendPolicy** takes current content of the *buffer*, recommended **NxtBc** and block-insertion times vector $\vec{\tau}_{\text{state}}$, and outputs the vector containing blocks to be added to the state. To enforce more properties, the **ExtendPolicy** also gets the timed honest-input sequence $\vec{\mathcal{I}}_H^T$.

Additional properties of Extend Policy Liveness and chain quality. Liveness corresponds to updating the state at least once after some time. If the adversary does not propose blocks, the **ExtendPolicy** will propose. The other property forces a tag to every block proposal, which indicates whether it comes from a honest party or not. Using that property, chain quality is ensured by first, ensuring that there are frequent enough proposal from the honest party, and second, that such proposals fulfil some quality properties. If these requirements are not met, the ledger will define and add a default block to the state.

The common prefix property, which guarantees that blocks that have made it deep enough in the blockchain of a honest miner, will eventually make it into the blockchain of every honest miner, can not be guaranteed in reality. This is because we can not guarantee, that all miners see exactly the same blockchain length. For an example, the network delays. Because it is unclear, how to define a state, which all parties would have the same view of at any point, the relaxation of interpreting state as the state of the party with the longest blockchain. The adversary is allowed to define for every miner p_i, subchain state_i of *state*, with length pt_i. The adversary is only allowed to move the pointer forward. Also, the adversary must not define pointers, which are further apart than $windoSize < N$. *windoSize* is one of the ledgers core parameters. The sliding window describes the advancing of the pointers with the advancing of the window head with the *state*.

Desynchronized miners Miners, who have recently registered or registered parties, that get de-registered from the clock, are called desynchronized, and the set containing all desynchronized miners is denoted $\mathcal{P}_{DS}$. By assuming a communication delay of up to Δ for a NEW-MINER message to reach others and up to Δ for response, the new miner can be in the control of the adversary for up to $Delay = 2 \cdot \Delta$ rounds.

4.2 Ledger functionality

The functionality is parametrized by the algorithms, **Validate**, **ExtendPolicy**, **Blockify** and **predict-time**, and two parameters: *windowSize* and $Delay \in N$. Other variables are initialized as:

$$\textbf{state} = \vec{\tau}_{state} = \textbf{NxtBc} = \epsilon$$

$$\textbf{buffer} = \emptyset, \tau_L = 0$$

The functionality maintains sets of registered parties $\mathcal{P}$, honest parties $\mathcal{H} \subseteq \mathcal{P}$ and de-synchronized parties $\mathcal{P}_{DS} \subset \mathcal{H}$, all initially empty. As miners join the system, the joining time is recorded and they are added to the sets . $\mathcal{P}$ and $\mathcal{H}$. If the time is greater than 0, the party is also added to the set $\mathcal{P}_{DS}$. The functionality maintains for every party $p_i \in \mathcal{P}$ a pointer pt_i (initially 1) and a current-state view $\text{state}_i := \epsilon$. It also keeps track of the timed honest-input sequence $\vec{\mathcal{I}}_H^T := \epsilon$.

Upon receiving input I First requests current time from $\mathcal{G}_{\text{clock}}$ and based on the response (τ) sets $\tau_L := \tau$

1. Let $\hat{\mathcal{P}} \subseteq \mathcal{P}_{\text{DS}}$, be the set of parties that have been registered ($\tau' < \tau_L - Delay$). Set $\mathcal{P}_{\text{DS}} := \mathcal{P}_{\text{DS}} \setminus \hat{\mathcal{P}}$.

2. If input was received from an honest party $p_i \in \mathcal{P}$:

   a Set $\vec{\mathcal{I}}_H^T := \vec{\mathcal{I}}_H^T || (I, p_i, \tau_L)$.

   b Compute

$$\vec{N} = (\vec{N}_1, ..., \vec{N}_l) := \textbf{ExtendPolicy}(\vec{\mathcal{I}}_H^T, \text{state}, \textbf{NxtBc}, \textbf{buffer}, \vec{\tau}_{state})$$

and if $\vec{N} \neq \epsilon$ set

$$\mathbf{state} = \mathbf{state}||\mathbf{Blockify}(\vec{N}_1)||...||\mathbf{Blockify}(\vec{N}_l)$$

and $\vec{\tau}_{state} = \vec{\tau}_{state}||\tau_L^l$, where $\tau_L^l = \tau_L||...||\tau_L$.

c For each $BTX \in$ **buffer**, if BTX invalid, remove from **buffer** and set **NxtBc** $= \epsilon$.

d If for some miner in the set of honest miners, who is not desynchronized, the difference between their pointer pt_i and the length of current longest blockchain is greater than the *windowSize*, then update the pointers of all synchronized honest miners to the current longest blockchain.

3. Depending on the input, one of the following actions is performed:
• Submitting a transaction - a unique transaction id is picked, the transaction is validated and if valid added to the *buffer*. Finally informs the adversary of the new transaction.
• Reading the state - If the requester is a party $p_i \in \mathcal{P}$, responds with a prefix of the state without the last blocks depending on the pointer for this party. If the requestor in $\mathcal{A}$, respond the whole *state*, *buffer* and timed honest input sequence.
• Maintaing the ledger state - If the receiver is a honest party, then after updating the timed honest input sequence, the *predict-time* responds the next time send a *clock update* request. Otherwise send input to adversary.
• Adversary proposing the next block - First checks if the transactions are in the buffer, and upon verifying adds the transaction to a queue. Then creates a next block candidate with the transactions in the queue. Responds to the adversary with ok.
• Adversary setting state-slackness - Verifies, that each proposed pointer points after the known state length and that it is within the last *windowSize* state blocks. If so, then update the pointer. Otherwise set the pointer to point at the end of the *state*.
• Adversary setting the state for desynchronised party - the adversary can set the states for desynchronised parties.

5 Connecting the Model and the Ledger Protocol

In section 4. of the original paper, the main theorem, which means that under appropriate assumptions, Bitcoin realizes an instantiation of the ledger functionality, is proved. The Bitcoin protocol is cast as a UC protocol. First, the UC protocol, denoted as Ledger-Protocol, will be described, abstracting the components of Bitcoin which are relevant for constructing the ledger. Then the ledger functionality, implemented by the UC ledger protocol, is specified. Finally, the main theorem is stated and proved.

Informal theorem Fixes Validate and windowSize. States that for appropriate ExtendPolicy and for any function Blockify, the Ledger-Protocol instantiated with Validate and Blockify, securely realizes a ledger functionality. Some assumptions on network delays and mining power are done. mining pwer is roughly the ability to find proof of work via queries to random oracle. It states that, in any round, the adversary's mining power with desynchronized honest miners and corrupted miners does not exceed the mining power of honest miners. Also, there is an upper bound Δ on the maximum network delay.

The theorem is proved with a modularization of the Bitcoin protocol. The modularization is supposed to distill out a reactive state-extend subprocess, which will the lottery deciding the next state extending miner and propagating the new state to other miners. Lemma 1 is used to show, that the extend sub-process implements an appropriate reactive UC functionality $\mathcal{F}_{STX}$. Additionally, in the original paper, it is shown how some previous protocols can be cast as special cases of the new protocol, which allows comparing the different models and assumptions.

5.1 Bitcoin ledger as a UC Protocol

The following section is supposed to provide a formal desription of protocol Ledger-Protocol. The protocol assumes as hybrids the multi-cast network $\mathcal{F}_{N-MC}$ and a random oracle functionality $\mathcal{F}_{RO}$.

Additional terminology A blockchain $C = \mathbf{B}_1, ..., \mathbf{B}_n$ is a finite sequence of blocks. A block $\mathbf{B}_i = \langle s_i, st_i, n_i \rangle$ is a triple containing pointer s_i, state block st_i and a nonce n_i. The first block in the chain is the genesis block $\mathbf{G}$ containing only the genesis state $gen = \epsilon$. The head of the chain is denoted with $head(C) := \mathbf{B}_n$ and the length of the chain is the number of blocks in the chain. The underlying sequence of state blocks forms the state vector $\vec{st} := st_1||...||st_n$. It is suggested to think of the blockchain as of an encoding of the underlying state vector.

To capture blockchains with syntactically different state encodings, an algorithm *blockify* is used to map a vector of transactions into a state. Therefore every state (except the genesis state) has the following form $st = Blockify(\vec{N})$, where $\vec{N}$ is a vector of transactions. The validity of a blockchain depends on two aspects: chain-level validity (also know as syntactic validity) and a state-level validity (semantic validity). Syntactic validity is defined with respect to a difficulty parameter $D \in [2^\kappa]$, where κ is the security parameter. There is also a given hash function $H(\cdot) : \{0,1\}^* \Rightarrow \{0,1\}^\kappa$. it is required that for each $i > 1$, $s_i \in \mathbf{B}_i$ $s_i = H[\mathbf{B}_{i-1}]$ and $H[\mathbf{B}_i] < D$.
Semantic validity is defined on the state $\vec{st}$ encoded in the blockchain and specifies whether the content is valid. The Validate predicate of the Ledger functionality played a similar role. The semantic validity of the blockchain can be

defined using an algorithm *isvalidstate*, which builds upon Validate. For any choice of Validate, the blockchain protocol using *isvalidstate* implements the ledger parametrized with Validate. This Validate (from here on denoted as $ValidTx_B$) predicate ignores all other information besides state and transaction, that are being validated. *isvalidstate* checks that the blockchain can be built in an iterative manner, starting with the genesis state, and that each following transaction is valid. Also ensures that the state blocks contain a coinbase transaction, which assigns it to a miner. Following this, *isvalidchain* is used to denote a predicate, that is true iff the chain satisfies syntactic and semantic validity.

The Ledger Protocol The following will give the definition of the blockchain protocol $\textbf{Ledger-Protocol}_{q,D,T}$. Using a multicast network $\mathcal{F}_{N-MC}$, the protocol lets an arbitrary number of miners to communicate also enabling the adversary to send different messages to different parties. Miners are allowed to dynamically join and leave, and upon doing so, they register/deregister with the functionalities. Each party contains their own local blockchain, initially containing the genesis block, and allowed to differ from the blockchains of other miners. New blocks are added by first collecting valid transactions according to $ValidTx_B$,. then creating a block *st* using blockify_b and then trying to mine a new block. The mining is done using an algorithm, which takes as input the current chain, the new state and q, a number of attempts, then tries to find a proof-of-work, which would allow to extend the chain with a block encoding *st*. After each mining attempt, the current chain will be multicast. Upon receiving a longer chain, the received chain will be accepted as the new chain. The response to querying the state of the ledger, the longest chain is responded, with the most recent T blocks chopped off, to ensure output of a consistent ledger state.

To formally handle activations in UC, the rounds from glock $\mathcal{G}_{\mathcal{CLOCK}}$ are split into two mini rounds. The clock is in round r, if the current time of the clock is $\tau \in \{2r-1, 2r\}$. This also ensures that messages sent in the end of a round are received in the beginning of the next round. The first mini round is used to mine new blocks. The second mini round is used to fetch messages.

5.2 The Bitcoin Ledger

Next, it is shown how to instantiate the ledger functionality with appropriate parameters, so that it is implement by protocol **Ledger-Protocol**. To do this, specific instantiations for **Validate**, **Blockify**, **ExtendPolicy** and **predict-time** must be given. First, **predict-time** is defined as the predict-time protocol of the ledger protocol. The **Validate** will be the same predicate as the protocol uses - one that makes decisions based on only the **state** and **tx** ($Validate\left(\left(\mathbf{tx}, txid, \tau_L, p_i\right), \mathbf{state}, \mathbf{buffer}\right) := ValidTx_B\left(\mathbf{tx}, \mathbf{state}\right)$). **Blockify** can be an arbitrary algorithm, but for the proof to go through, the **Ledger-Protocol** must use the same algorithm. However, in definition 2 of the original paper, it is discussed that a meaningful **Blockify** should be in a relationship with **Validate**. The **ExtendPolicy** receives a list of block candidates and then, for each block, it verifies that the block is valid with respect to the state it extends. Moreover, it is said to ensure the following properties:

1. Minimal chain growth - the speed of the ledger is not too slow. Implemented by defining an upper bound $\mathbf{maxTime}_{window}$ within which at least **windowSize** state blocks have to be added during the interval.

2. The speed of the ledger is not too fast. Implemented by defining a lower bound $\mathbf{minTime}_{window}$, such that the adversary is not allowed to propose any more blocks if **windowSize** state blocks have to be added during the interval.

3. Chain quality - the adversary can not create too many blocks (with arbitrary but valid contents), implemented by defining an upper bound η on the number of adversial blocks within a sequence of state blocks.

4. If a transaction is 'old enough' and is still valid, it will eventually be added into the state.

These properties are enforced by the **ExtendPolicy** by first defining alternative blocks, which satisfy all of these properties. If the adversary gets caught trying to violate the properties, one of the previously generated blocks is proposed instead. In case of minimal chain growth violation, a sequence of previously generated blocks are proposed.

Definition 2 [1] A co-design of **Blockify** and **Validate** is non-self-disqualifying if there exists an efficiently computable function Dec mapping outputs of **Blockify** to vectors $\vec{N}$ such that there exists a validate predicate **Validate'** for which the following properties hold for any possible state $state = st_1||...||st_l$, buffer **buffer**, vectors $\vec{N} := (tx_1, ..., tx_m)$, and transaction tx:
1. $\mathbf{Validate}(tx, \mathbf{state}, \mathbf{buffer}) = \mathbf{Validate'}(tx, Dec(st_1)||..||Dec(st_l), \mathbf{buffer})$
2. $\mathbf{Validate}(tx, \mathbf{state}||\mathbf{Blockify}(\vec{N}), \mathbf{buffer}) =$
$\mathbf{Validate'}(tx, Dec(st_1)||..||Dec(st_l)||\vec{N}, \mathbf{buffer})$

5.3 Security Analysis

In this section, an alternative, modular description of the Ledger-Protocol is devised, called **Modular-Ledger-Protocol**. An ideal functionality $\mathcal{F}_{StX}$, which covers the state exchange subprocess in the Ledger Protocol, is defined. It is then proved, that the Modular-Ledger-Protocol, which uses invocations to $\mathcal{F}_{StX}$ implements the Bitcoin ledger previously described.

First, the ideal functionality $\mathcal{F}_{StX}$ is defined. This functionality allows parties to submit ledger states. The functionality then chooses, which states are accepted. The accepted states are sent to all parties in the system.

The ideal functionality $\mathcal{F}_{StX}$ has parameters Δ, the maximum delay an adversary can impose on a message, p_H the probability that a honest party's valid state is accepted and p_A the probability that the adversary's valid state is accepted. The Modular-Ledger-Protocol is said to use the same hybrids as Ledger-Protocol but the lottery implemented by the mining process by making calls to the ideal functionality. The Modular-Ledger-Protocol keeps the chopoff parameter T. Next, they provide Lemma 1.

Lemma 1 The blockchain protocol Ledger-Protocol$_{q,D,T}$ UC emulates protocol Modular-Ledger-Protocol$_T$ that runs in a hybrid world with access to the functionality $\mathcal{F}_{StX}^{\Delta,p_H,p_A}$ with $p_A := D/2^{\kappa}$ and $p_H := 1-(1-p_A)^q$, and where Δ denotes the upper bound on the network delay.[1, Lemma 1]

Then some properties of the Bitcoin protocol execution are captured. First, R is used to denote the number of rounds of a given protocol execution. Then the query power in an execution is captured. This can be captured for a logical round, in this case captured by a function $T_{QP}(r)$ (where r is the round) and the total query power of that round being $Q_{t_{rc}}^{r'} = \sum_{r=r'}^{r'+t_{rc}-1} T_{qp}(r)$. In here t_{rc} denotes an interval of rounds. They also calculate the total number of activations the honest parties get in a round $q_H^{(r)} = \sum_{p_i honestinroundr} q_i^{(r)}$, where $q_i^{(r)}$ is the number of activations a miner gets in round r. Denoted by $q_A^{(r)}$, the number of queries the adversary makes to the ideal functionality in round r. The total query power: $T_{qp}(r) = q_A^{(r)} + q_H^{(r)}$.

By adding in the probabilities of mining a block, the total mining power per round can be found by: $T_{mp}(r) := q_A^{(r)} \cdot p_A + q_H^{(r)} \cdot p_H$. When quantifying adversarial mining power in an execution, the desynchronised miners must also be taken into consideration. The equation is as follows: $mp_A(r) := p_A \cdot q_A^{(r)} + p_H \cdot \sum_{p_i isdesynchronised} q_i^{(r)}$. They also assume that there is a parameter $\rho \in (0,1)$ such that in any round r the relation $mp_A(r) \leq \rho \cdot T_{mp}(r)$ holds.

They then define

$$\beta_r := \rho \cdot T_{mp}(r).$$

The probability that a miner is successful at mining at least one block and extending the state is $\alpha_{i,r} := 1-(1-p_H)^{q_{i,r}}$, where $q_{i,r}$ is the number of activations party i gets in round r. The probability that at least one honest synchronised miner gets to extend the state is

$$\alpha_r := 1 - \Pi_{honestsyncp_i}(1-\alpha_{i,r}) = 1 - \Pi_{honestsyncp_i}(1-p_H)^{q_{i,r}}.$$

By taking the worst-case into analysis,

$$\alpha := \min\{\alpha_r\}_{r\in[R]}, \text{ and } \beta := \max\{\beta_r\}_{r\in[R]}$$

They then proceed to with the theorem. In this theorem they instantiate Modular-Ledger-Protocol, give a relation involving α and β, and if this relation holds, the Modular-Ledger-Protocol UC realises the Ledger for any ledger parameters in given ranges. The ranges are then given as functions related to T, Δ, α, β and equations using these with additionally the round.

6 Real world implementation

The ledger proposed in the paper can be used as a black-box and can help with security analysis of the complete system. In case of the Bitcoin implementation, each account has a key pair. The public key is hashed and the hash is then used as the account id. The key pair is used to sign transactions. The receiver of a transaction, has to verify both the validity of the signature and the validity of the account id. The ledger which is given in the paper, is said to be an abstraction of the Bitcoin algorithm. Also the uniqueness of accounts is realized. Finally, transaction liveness is discussed. A single valid transaction is assumed to be contained in the state within the next $3 * windowSize$ time from submitting.

7 Conclusion

In the paper, the original authors modeled the Bitcoin protocol, described the ideal functionalities, instantiated the Bitcoin ledger protocol in a hybrid world and using UC composition theorem showed that a modular ledger protocol UC realised the original ledger protocol. They then proved a theorem, which gives ranges for the ledger parameters. They concluded, that the ledger protocol executed in the hybrid world UC realizes their described Bitcoin ledger functionality with respect to parameters from Theorem 1. Finally they covered some real world implementation details.

Acknowledgement

I would like to thank Ehsan Ebrahimi for his guidance and advice throughout the project.

References

[1] C. Badertscher, U. Maurer, D. Tschudi, and V. Zikas, “Bitcoin as a transaction ledger: A composable treatment.” 37th International Cryptology Conference, 2018. Extended version.

[2] J. A. Garay, A. Kiayias, and N. Leonardos, “The bitcoin backbone protocol: Analysis and applications.” Elisabeth Oswald and Marc Fischlin, editors,

EUROCRYPT 2015, Part II, volume 9057 of LNCS, pages 281–310, April 2015.

[3] R. Pass, L. Seeman, and A. Shelat, “Analysis of the blockchain protocol in asynchronous networks.” Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part II, pages 643–673, 2017.

[4] R. Canetti, Y. Dodis, R. Pass, and S. Walfish, “Universally composable security with global setup.” In Salil P. Vadhan, editor, TCC 2007, volume 4392 of LNCS, pages 61–85, 2007.

[5] R. Canetti, “Universally composable security: A new paradigm for cryptographic protocols,” 2001. An updated version of ‘A unified framework for analyzing security of protocols’. Later updated in January 2005, December 2005 and July 2013.

[6] J. Katz, U. Maurer, B. Tackmann, and V. Zikas, “Universally composable synchronous computation.” Amit Sahai, editor, TCC 2013, volume 7785 of LNCS, pages 477–498, March 2013.

[7] A. Kiayias, H.-S. Zhou, and V. Zikas, “Fair and robust multi-party computation using a global transaction ledger.” Marc Fischlin and Jean-Sebastien Coron, editors, EUROCRYPT 2016, Part II, volume 9666 of LNCS, pages 705–734, 2016.