Formal analysis of Bitcoin-like protocol

zondag 12 oktober 2014
21:22

Formal analysis of Bitcoin-like protocols

The original paper does not contain formal proofs of the security of Bitcoin.
A recent paper by Garay, Kiayias and Leonardos attempts to do so: https://eprint.iacr.org/2014/765.pdf
These notes are based on their paper, I recommend reading it.
They define the blockchain process (public blockchain, mining, fork resolving),
which they call the Bitcoin Backbone protocol, and provide security proofs in
a somewhat idealized setting, called the q-bounded synchronous setting.
They have abstracted the application of Bitcoin by yet-undefined functions V(), R() and I()
that are used by the Bitcoin Backbone protocol to verify, output and contribute to the blockchain, respectively.

Model and definitions

The q-bounded synchronous setting

There are a fixed number n of nodes in network,
the number n is considered unknown and the protocol may not depend on this value.
The network communication graph is not fully connected, but messages are forwarded to all (P2P).
Thus each node can issue a BROADCAST command that sends a message via the P2P-network.
Also, nodes can receive INPUT() (e.g., user generated transaction) and read network messages via RECEIVE().
The network is synchronous, i.e., in 1 round everybody may broadcast 1 message to all which is received by all by the start of the next round.
The rounds are thus defined in network connectivity. E.g., in the Bitcoin network it has been measured that it takes about 3 seconds for a BROADCAST to reach every node, one can thus think of as 1 round lasting 3 seconds.

The Adversary is adaptive (i.e., it can take control of nodes on the fly under the given bounds), as well as rushing (i.e., it can see all other messages first before it sends its own).
The number of nodes cannot be depended on as source information of messages can be spoofed, to model this, the adversary is allowed to alter source information on every message.
However, the adversary is restricted from altering contents of honest nodes, nor block their messages.
Do note that adversary can abuse BROADCAST and send different messages to different nodes.

All nodes have access to a global Oracle H(.) and are allowed to perform a number of queries q per round.
So the hashing power of all nodes are assumed equal, but 'real world' cases of varying hashing power can be seen as parties having different number of nodes under their control.
The adversary is allowed $t*q$ queries, where $t < n$ is the number of corrupted nodes.

Blockchain notation

Let G(.), H(.) be cryptographic hash functions with output in $\{0,1\}^\kappa$. (Imagine both be SHA-256.)

A *block* is a triple of the form $B =< s, x, ctr >$, where $s \in \{0,1\}^\kappa, x \in \{0,1\}^*, ctr \in \{0,1\}^\ell$ satisfying predicate $validBlock^D(B)$:

$\left(H(ctr, G(s,x)) \leq D\right)$

Parameter $D \in \mathbb{N}$ is the *difficulty level* and parameter ℓ with $q \leq 2^l$ is to ensure ctr is suitably short, e.g., $\ell = 32$.

The restrictions on ctr are a bit arbitrary, in general one can restrict ctr to any subset of all bitstrings as long as it has at least q elements.

Note that lower D will actually be more difficult, the probability of success is $D/2^\kappa$.

Here s will form the link to the preceding block in the chain, x will be the added content to the chain, and ctr will provide the freedom in finding a block satisfying the predicate $validBlock^D(B)$.

Blockchain

- A *blockchain* $\mathcal{C}$ is a sequence of blocks. Its rightmost block is the head or end of the chain, denoted $head(\mathcal{C})$.
- An empty string ε is also a chain, by convention: $head(\varepsilon) = \varepsilon$.
- A chain $\mathcal{C}$ with $head(\mathcal{C}) =< s', x', r' >$ can be extended to a longer chain by appending a valid block $B =< s, x, r >$ with $s = H(r', G(s', x'))$.
  For $\mathcal{C} = \varepsilon$, any block may extend it, i.e., no restriction on s, e.g., set $s = 0$.
  Then the extended chain $\mathcal{C}_{new} = \mathcal{C}B$ has $head(\mathcal{C}_{new}) = B$.
- The *length* of a chain $len(\mathcal{C})$ is the number of blocks.
- For chain $\mathcal{C}$ of length m, for any $k \in \mathbb{N}^0$, we denote $\mathcal{C}^{\lceil k}$ the chain by removing k times its head, i.e., pruning the k rightmost blocks.
- For $k \geq len(\mathcal{C})$, $\mathcal{C}^{\lceil k} = \varepsilon$.
- If $\mathcal{C}_1$ is a prefix of $\mathcal{C}_2$, i.e., $\exists k \in \mathbb{N}^0 \left[\mathcal{C}_1 = \mathcal{C}_2^{\lceil k}\right]$, we denote $\mathcal{C}_1 \preccurlyeq \mathcal{C}_2$.

Bitcoin has chains of variable difficulty, i.e., D changes across different blocks,
which is determined by the contents of the chain up to that block (which includes time stamps).
In their analysis, as the number of nodes as well as the hashing power per node per round is fixed, also a fixed difficulty D is assumed.

Bitcoin Backbone protocol

The Bitcoin Backbone protocol is defined with the following functions.
They depend on application specific functions $V(\cdot), R(\cdot)$ and $I(\cdot)$:

- $V(x_{\mathcal{C}})$ returns true if the vector of block contents $x_1, \ldots, x_m$, $x_i \in \{0,1\}^*$, forms a valid vector in the application setting, e.g., Bitcoin: the contents forms a valid ledger of valid transactions. And returns false otherwise.
  The Bitcoin Backbone protocol will use this function to confirm whether a Blockchain is valid.
- $R(x_{\mathcal{C}})$ returns the vector of block contents in an application specific formatted form, this is how the application on top of the Bitcoin Backbone protocol can access the content in the Blockchain.
- $I(st, \mathcal{C}, round, userinput, networkinput)$ returns a pair (st', x), where st and st' are the old and updated state, respectively. I further receives the current Blockchain, user-input as well as network-input.
  The Bitcoin Backbone protocol calls this application dependent function to determine the contents $x \in \{0,1\}^*$ to put into the block which it will try to extend the Blockchain with.

This function verifies whether a chain of blocks is valid:

Function Validate($\mathcal{C}$)

1. $b \leftarrow True$
2. $x_{\mathcal{C}} \leftarrow < \varepsilon >$ (empty vector)
3. **while** $\mathcal{C} \neq \varepsilon$ **do**
4. $< s, x, ctr > \leftarrow head(\mathcal{C})$
5. **if** $validBlock^D(< s, x, ctr >)$ **then**
6. $x_{\mathcal{C}} \leftarrow append(x_{\mathcal{C}}, x)$

7. $\mathcal{C} \leftarrow \mathcal{C}^{\lceil 1}$
8. **if** $\mathcal{C} \neq \varepsilon$ **then**
9. $< s', x', ctr' > \leftarrow head(\mathcal{C})$
10. $b \leftarrow b \wedge \left(s = H\left(ctr', G(s', x')\right)\right)$
11. **end if**
12. **else**
13. $b \leftarrow False$
14. **end if**
15. **end while**
16. **return** $b \wedge V(x_{\mathcal{C}})$

Note that their paper actually contains some errors:
There is no break and without removing the head like in step 6, the function will be caught in an endless loop.
Also, they don't actually verify whether the value s is correct.

This function returns the 'largest' blockchain from a set.
Function maxvalid$(\mathcal{C}_1, \ldots, \mathcal{C}_k)$

1. $temp \leftarrow \varepsilon$
2. **for** $i = 1$ **to** k **do**
3. **if** $validate(\mathcal{C}_i)$ **then**
4. $temp \leftarrow \max(temp, \mathcal{C}_i)$
5. **end if**
6. **end for**
7. **return** $temp$

This function defines the comparison (which of two blockchains is 'larger'):
Function max$(\mathcal{C}_1, \mathcal{C}_2)$

1. $temp \leftarrow \mathcal{C}_1$
2. **if** $len(\mathcal{C}_2) > len(\mathcal{C}_1)$ **then**
3. $temp \leftarrow \mathcal{C}_2$
4. **end if**
5. **return** $temp$

Other options exist for tie-breaking in case of equal length: lexicographic order, random picking.
Note that Bitcoin actually uses a different definition of 'length' within this function and for a very good reason,
however in the q-bounded synchronous setting they are essentially equivalent.

This function attempts to solve the proof-of-work and extend the chain with a block with contents x, otherwise it just returns the input chain.
Function pow$(x, \mathcal{C})$

1. **if** $\mathcal{C} = \varepsilon$ **then**
2. $\tilde{s} \leftarrow 0$
3. **else**
4. $< \tilde{s}, \tilde{x}, \tilde{r} > \leftarrow head(\mathcal{C})$
5. **end if**
6. $h \leftarrow G(\tilde{s}, x)$
7. **for** $ctr \in \{0,1\}^{\ell}$ **do**
8. **if** $H(ctr, h) \leq D$ **then**
9. $B \leftarrow < \tilde{s}, x, ctr >$
10. $\mathcal{C} \leftarrow \mathcal{C}B$
11. **break**
12. **end if**
13. **end for**

14. **return** $\mathcal{C}$

The following Bitcoin Backbone algorithm is run by all nodes indefinitely:

Bitcoin Backbone protocol

1. $\mathcal{C} \leftarrow \varepsilon$
2. $st \leftarrow \varepsilon$
3. $round \leftarrow 0$
4. **while** $True$ **do**
5. $\tilde{\mathcal{C}} \leftarrow maxvalid(\mathcal{C}, \text{any chain } \mathcal{C}' \text{found in RECEIVE}())$
6. $< st, x > \leftarrow I\left(st, \tilde{\mathcal{C}}, round, INPUT(), RECEIVE()\right)$
7. $\mathcal{C}_{new} \leftarrow pow(x, \tilde{\mathcal{C}})$
8. **if** $\mathcal{C} \neq \mathcal{C}_{new}$ **then**
9. $\mathcal{C} \leftarrow \mathcal{C}_{new}$
10. $BROADCAST(\mathcal{C})$
11. **end if**
12. $round \leftarrow round + 1$
13. **if** $INPUT()$ contains READ **then**
14. OUTPUT($R(x_{\mathcal{C}})$)
15. **end if**
16. **end while**

Step 5 determines the longest known chain which it will try to extend
Step 6 determines the content of the block it will try to append
Step 7 attempts the proof of work
Step 8 broadcasts any update in the chain in that round (e.g., via RECEIVE() or pow())
Step 13-15 allows the application to query the contents of the blockchain
Also note that every node that runs the protocol maintains its own version of the chain that may differ between nodes.

Protocol security properties

Definition Common Prefix Property

The protocol has the Common Prefix Property with parameter $k \in \mathbb{N}$
If for any pair of honest players P_1, P_2 that maintain chains $\mathcal{C}_1, \mathcal{C}_2$ it holds that

$$\mathcal{C}_1^{\lceil k} \preccurlyeq \mathcal{C}_2 \quad \text{and} \quad \mathcal{C}_2^{\lceil k} \preccurlyeq \mathcal{C}_1$$

In other words, the common prefix property ensures that blocks at depth k or deeper are definitively agreed upon between all honest players, and cannot be changed anymore.
This means that at some points blocks become 'written in stone', but doesn't mean they will contain meaningful information (e.g., a very powerful adversary may cause all blocks to be empty, preventing any transactions from occuring). The authors therefore consider the following property:

Definition Chain Quality Property

The protocol has the Chain Quality Property with parameters $\mu \in \mathbb{R}, l \in \mathbb{N}$
if for any honest party P with chain $\mathcal{C}$ it holds that
For all l consecutive blocks in $\mathcal{C}$, the ratio of adversarial blocks it at most μ.

In other words, the chain quality property guarantees that information by honest parties will eventually be part of the blockchain.

Notation

Let $\alpha = (n - t) * q * \frac{D}{2^\kappa}, \beta = t * q * \frac{D}{2^\kappa}, \gamma = \alpha - \alpha^2, f = \alpha + \beta$
α, β are the upper bounds on the expected number of solutions that the honest parties / adversary can compute in one round.

The ratio α/β denotes the ratio in computer power between honest parties and the adversary. When α is small (e.g., when f is small), then $\gamma \approx \alpha$.

To put some things in perspective, it takes about 3 seconds for a message to reach all nodes in Bitcoin, so assume a round lasts 3 seconds, while on average a solution is found every 10 minutes. Then $f = \frac{3}{10*60} = 0.005$, and $\gamma \approx \alpha$.

Theorem 1 (CPP)

Assume $f < 1$ and $\gamma \geq (1+\delta) * \lambda * \beta$, for some $\delta \in (0,1)$ and $\lambda \geq 1$ such that $\lambda^2 - f\lambda - 1 \geq 0$.
Let $\mathcal{S}$ be the set of chains of the honest parties at a given round of the protocol.
Then the probability that $\mathcal{S}$ does not satisfy the Common Prefix Property with parameter k is at most $e^{-\Omega(\delta^3 k)}$

Proof

See the paper

Discussion

Less precisely, if $\gamma > \lambda * \beta$ for $\lambda \geq 1$ with $\lambda^2 - f\lambda + 1 \geq 0$, then the probability that the chains of the honest parties do not satisfy the CPP with parameter k drops exponentially in k.
If f is very close to 0, i.e., high network synchronicity, then λ can be chosen very close to 1 and thus establish the CPP as long as an honest majority of nodes is guaranteed, i.e., the adversary controls strictly less than 50% of the hashing power.
On the other hand, when the network has low synchronicity and f gets closer to 1, achieving a CPP requires $\lambda \to \phi = (1+\sqrt{5})/2$, which sets much stricter bounds on the adversary.

Theorem 2 (CQP)

Assume $f < 1$ and $\gamma \geq (1+\delta) * \lambda * \beta$ for some $\delta \in (0,1)$.
For any honest player P with chain $\mathcal{C}$, and any l consecutive blocks in $\mathcal{C}$,
the probability that the adversary contributed more than $\frac{(1-\delta/3)*l}{\lambda}$ of these blocks is less than $e^{-\Omega(\delta^2 l)}$.

Proof

See the paper

Discussion

Less precisely, if $\gamma > \lambda * \beta$ for some $\lambda \geq 1$ then the ratio of blocks contributed by honest players in the chain of any honest player is at least $\left(1 - \frac{1}{\lambda}\right)$. If λ is close to 1, we obtain that the blockchain maintained by honest players is guaranteed to have few, but still some, blocks contributed by honest players.
Selfish mining indeed allows the adversary to achieve this ratio.

On top of the Bitcoin Backbone protocol actually various kinds of applications can be implemented. In particular, we will look at a simple Byzantine Agreement protocol as well as a Bitcoin-like Public Ledger protocol.

Byzantine Agreement

A number of parties $P_1, \dots, P_n$ each having their own input $v_i \in \{0,1\}$.
A protocol solves Byzantine Agreement in the q-bounded synchronous setting if it satisfies the 2 properties:

- Agreement: There is a round after which all honest parties return the same output.
- Validity: The output returned by an honest party equals the input of a honest party that has not been non-corrupted till the end.

A solution against (1/3)-bounded adversaries, i.e., $t < n/3$, is instantiated upon the Bitcoin Backbone

protocol as follows:

- $V(\cdot)$: input validation: $V(< x_1, \dots, x_m >) = True$ iff $x_i = (v_i, p_i)$ with $v_i \in \{0,1\}, p_i \in \{0,1\}^\kappa$ for all i.
- $R(\cdot)$: chain reading: If $V(< x_1, \dots, x_m >) = True$ and $m \geq 2k$, return the majority bit of $v_1, \dots, v_k$. Otherwise return $\perp$.
- $I(\cdot)$: input contribution: $I(st, \mathcal{C}, round, INPUT(\,))$ returns (v, p), where v is the parties original input and p is a random κ-bit string.

I.e., each node tries to extend the blockchain with a block containing its own private input bit.
The protocol is run at least $2\,k$ rounds, aiming for the CPP with parameter k, meaning all honest players should agree on at least the first $2\,k - k = k$ blocks.
It aims at the CQP with parameters $l = k$ and $\mu = \frac{1}{2}$, such that of those first k blocks the adversary has no majority. The majority bit thus has to be equal to some honest players input.

Lemma Agreement

Suppose $f < 1$ and $\gamma \geq (1+\delta) * 2 * \beta$, for $\delta \in (0,1)$. Then this BA-protocol satisfies Agreement in $O(k)$ rounds with probability at least $1 - e^{-\Omega(\delta^3 k)}$.

Proof

In order for Agreement to be violated, at least two honest parties should have upon termination chains $\mathcal{C}_1, \mathcal{C}_2$ such that $\mathcal{C}_1^{\lceil k} \neq \mathcal{C}_2^{\lceil k}$. In particular, the set of chains $\{\mathcal{C}_1, \mathcal{C}_2\}$ belonging to honest parties does not satisfy the CPP. The lemma follows directly from Theorem 1 (CPP). ■

Lemma Validity

Suppose $f < 1$ and $\gamma \geq (1+\delta) * 2 * \beta$, for $\delta \in (0,1)$. Then this BA-protocol satisfies Validity in $O(k)$ rounds with probability $1 - e^{-\Omega(\delta^2 k)}$.

Proof

For the property to be satisfied we only need to ensure that in $\mathcal{C}$, the chain from any honest party upon termination, the majority of the inputs $v_1, \dots, v_k$ was computed by the honest parties.
Theorem 2 with $\lambda = 2$ and $l = k$ provides this:
$\frac{(1-\delta/3) * l}{\lambda} < \frac{k}{\lambda} = \frac{k}{2}$.
$\lambda = 2$ implies that security is limited to $(1/3)$-bounded adversaries: $t < n/3$.
■

The paper actually also describes a more complicated Byzantine Agreement Protocol secure against a (1/2)-bounded adversary, i.e., $t < n/2$, for which I refer to their paper.

Public ledger

The paper also describes a Public Ledger application on top of the Bitcoin Backbone protocol, however it has abstracted the precise description of transactions and the ledger, and they have left out the mining reward.
They use the above theorems to prove security of the Public Ledger:
In particular, the CPP can be used to prove Persistence of a Public ledger, i.e., transactions are permanent and ordered against a (1/2)-bounded adversary.
Furthermore, the CQP can be used to prove Liveness of a Public ledger, i.e., transactions of honest parties are eventually included against a (1/2)-bounded adversary.

Let $\mathcal{T}$ be the set of valid transactions and $\mathcal{L}$ be the set of valid ledgers $x \in \mathcal{T}^*$.
A transaction $tx \in \mathcal{T}$ may be associated with one or more accounts $a_1, a_2, \dots$
The Bitcoin Backbone protocol process sequences of transactions into a block and subsequently in the blockchain. The ledger is the concatenation of the sequences of each block in the blockchain.
The global activity in the ledger is modeled in one single public stateful oracle $Txgen$ that will control a

set of accounts and issue transactions on their behalf:

- Function $GenAccount(1^\kappa)$: generates an account a
- Function $IssueTrans(1^\kappa, \widetilde{tx})$: generates a transaction tx provided that $\widetilde{tx}$ is a suitably formed string

Transactions tx_1, tx_2 are conflicting i.f.f. $C(tx_1, tx_2) = 1$.
Transactions need to be *unmalleable* into conflicting transactions, i.e., for any PPT $\mathcal{A}$ with access to $Txgen$ the probability that it produces a transaction tx' such that $C(tx', tx) = 1$ for some tx generated by $Txgen$ is negligible.
A transaction tx is *neutral* iff $\forall tx' \in \mathcal{T}[C(tx, tx') = 0]$, e.g., a nonce. These are used below for the sole purpose of ensuring independence between the POW instances for each node.
In Bitcoin this is actually achieved with the reward (i.e., each node uses a different reward destination address).

To use the Bitcoin Backbone protocol, we have to define three functions:

- $V(\cdot)$: input validation: $V(< x_1, \dots, x_m >) = True$ iff $< x_1, \dots, x_m > \in \mathcal{L}$.
- $R(\cdot)$: read chain: if $V(< x_1, \dots, x_m >) = True$, return $< x_1, \dots, x_m >$, otherwise return $\perp$.
- $I(\cdot)$: $I(st, \mathcal{C}, round, INPUT())$:
    1. Read to-be-processed-transactions $v =< tx_1, \dots, tx_m >$ from INPUT() and RECEIVE().
    2. Retain largest subsequence $v' \preccurlyeq v$ that is valid with respect to $x_{\mathcal{C}}$ (i.e., tx that do not conflict with any transaction in $x_{\mathcal{C}}$ and are not already present in $x_{\mathcal{C}}$)
    3. Return $x = tx_0 \mid v'$, where tx_0 is a random neutral nonce transaction.

This defines the protocol Π_{PL}.

Definition Persistence

The protocol has the Persistence property for some $k \in \mathbb{N}$ if in a certain round an honest player reports a ledger that contains a transaction tx in a block more than k blocks away from the end of the ledger, then tx will always be reported in the same position in the ledger by any honest players from this round on.

Definition Liveness

The protocol has the Liveness property for $u, k \in \mathbb{N}$ ('wait time' and 'depth') when, provided that an honest player transaction is reported for u consecutive rounds, there exists an honest party who will report this transaction at a block more than k blocks from the end of the ledger.

Definition Robust Public Transaction Ledger

A protocol Π implements a *robust public transaction ledger* in the q-bounded synchronous setting if it satisfies both Persistence and Liveness.

Lemma (Persistence)

Suppose $f < 1$ and $\gamma \geq (1 + \delta) * \lambda * \beta$, for $\delta \in (0,1)$ and $\lambda \geq 1$ and $\lambda^2 - f\lambda - 1 \geq 0$. Protocol Π_{PL} satisfies Persistence with parameter k with probability $1 - e^{-\Omega(\delta^3 k)}$.

Proof

See paper

Lemma (Liveness)

Suppose $f < 1$ and $\gamma \geq (1 + \delta) * \lambda * \beta$, for $\delta \in (0,1)$ and $\lambda \geq 1$. Let $k \in \mathbb{N}$ and assume transactions are *unmalleable* into conflicting transactions (see above). Then Π_{PL} satisfies Liveness with wait time $u = 2k/(1 - \delta)\gamma$ and depth parameter k with probability at least $1 - e^{-\Omega(\delta^2 k)}$.

Proof

See paper

The above lemma's essential allow the Public Ledger to be secure against a (1/2)-bounded adversary, i.e., $t < n/2$, when f is close to 0.

When the network has lower connectivity, or the difficulty is such that solutions are found more frequently, then f will be further away from 0.
And thus stricter bounds on the adversary are required for security.
In particular this affects the security of various alternate coins (Litecoin, etc.) that boast of faster transaction confirmation times.