**EE 374: Blockchain Foundations** **Stanford, Spring 2022**

# Lecture 11: Security in Earnest I

May 11

Lecturer: Dr. Dionysis Zindros Scribe: Solal Afota

## 1 The Simulation Environment

In order to rigorously define properties of our blockchain and give security proofs, we need to precisely define how we will simulate our environment: the setup and how each round happens.

**Algorithm 1** The environment and network model running for a polynomial number of rounds $\mathsf{poly}(\kappa)$.

$r \leftarrow 0$
**function** $\mathcal{Z}^{n,t}_{\Pi,\mathcal{A}}(1^\kappa)$
  $\mathcal{G} \xleftarrow{\$} \{0,1\}^\kappa$ ▷ Genesis block
  **for** $i \leftarrow 1$ to $n-t$ **do** ▷ Boot stateful honest parties
    $P_i \leftarrow \mathsf{new}\ \ \Pi^{H_{\kappa,i}}(\mathcal{G})$
  **end for**
  $A \leftarrow \mathsf{new}\ \ \mathcal{A}^{H_{\kappa,0}}(\mathcal{G}, n, t)$ ▷ Boot stateful adversary
  $\overline{M} \leftarrow []$ ▷ 2D array of messages
  **for** $i \leftarrow 1$ to $n-t$ **do**
    $\overline{M}[i] \leftarrow []$ ▷ Each honest party has an array of messages
  **end for**
  **while** $r < \mathsf{poly}(\kappa)$ **do** ▷ Number of rounds
    $r \leftarrow r + 1$
    $M \leftarrow \emptyset$
    **for** $i \leftarrow 1$ to $n-t$ **do** ▷ Execute honest party $i$ for round $r$
      $Q \leftarrow q$ ▷ Maximum number of oracle queries per honest party (Section 2)
      $M \leftarrow M \cup \{P_i.\mathsf{execute}^H(\overline{M}[i])\}$ ▷ Adversary collects all messages
    **end for**
    $Q \leftarrow tq$ ▷ Max number of Adversarial oracle queries
    $\overline{M} \leftarrow A.\mathsf{execute}^H(M)$ ▷ Execute rushing adversary for round $r$
    **for** $m \in M$ **do** ▷ Ensure all parties will receive message $m$
      **for** $i \leftarrow 1$ to $n-t$ **do**
        $\mathsf{assert}(m \in \overline{M}[i])$ ▷ Non-eclipsing assumption
      **end for**
    **end for**
  **end while**
**end function**

### 1.1 A Simplification: Quantize Time

Notice that we are running the simulation in rounds (line 12). In the real world, time is continuous, however by breaking down the simulation into short rounds, it makes it much easier to define and prove security properties of our blockchain. Furthermore, it makes it easier to define the properties of our adversary as seen below.

### 1.2 Rushing Adversary

In this environment, we are assuming a Rushing Adversary. This is because every round (lines 12-26), we first simulate the honest parties independently - they do not see the messages produced by each other that round - (lines 15-19), collect all the messages on the gossip network (line 17), then run the adversary with all the gossiped messages (line 20).

### 1.3 Sybil Adversary & Non-Eclipsing Assumption

The adversary sees the messages gossiped by each honest party before the other honest parties. The adversary then has the power to manipulate what messages the honest parties will see next round in the following way

1. The adversary can inject new messages
2. The adversary can reorder messages
3. The adversary can introduce disagreement
4. The adversary cannot censor messages (lines 20, 21, 22)

The fourth point is due to the Non-Eclipsing Assumption: since there is a path of honest parties between any two honest parties, and each honest party follows the algorithm detailed in section 3, we know that an honestly produced message will be propagated to all honest parties on the next round.

## 2 Random Oracle Model

In the simulation algorithm, for both the honest parties and adversarial parties we write $\mathsf{execute}^H$ (lines 17, 20). This means that we model the hash function as a random oracle and give both the honest and adversarial parties Black-Box access to the oracle. This means that for any "new" input, the output is queried uniformly at random from the output space (line 10) and returned. Furthermore, when an input is queried for the first time, it is stored in a cache (stored in $\mathcal{T}$), therefore if the same input is later queried, the value is looked up in the cache and returned (line 12). Black-Box access means that the parties do not have access to the cache or the random sampling algorithm. They can only submit a query $x$ and receive a response $\mathcal{T}[x]$.
Secondly, in order to model the hash rate, we give each party a maximum number of oracle queries per round. Each honest party receives $q$ queries, and the adversary receives $qt$ queries. (line 3, 9).

**Algorithm 2** The Hash Function in the Random Oracle Model

```
1: r ← 0
2: T ← {}                                              ▷ Initiate Cache
3: Q ← 0                                               ▷ q for honest parties, qt for adversary
4: function H_{κ,i}(x)
5:     if x ∉ T then                                   ▷ First time being queried
6:         if Q = 0 then                               ▷ Out of Queries
7:             return ⊥
8:         end if
9:         Q ← Q − 1
10:        T[x] ←$ {0,1}^κ   ▷ Sample uniformly at random from output space and store in Cache
11:    end if
12:    return T[x]                                     ▷ Return value from Cache
13: end function
```

## 3 Honest Party Algorithm

Below is a class of algorithms belonging to an honest party.

The first algorithm is a constructor.

The second algorithm is used to simulate every honest party that mines during each round of the protocol. In this simulation, every honest party follows the longest chain rule, at the beginning of each round they adopt the longest, valid chain (line 8). If they learn about a new chain that is longer then their current one, they gossip it (line 9,10). The honest party then tries to mine a block using the transactions in their mempool (line 12, 13). If a block is successfully mined, the honest party will gossip it.

The third algorithm is used to extract all transactions from a blockchain. This is useful when validating a new chain as we need to check all transactions starting from the genesis state to ensure that there are no invalid transactions and to maintain an up-to-date UTXO set.

**Algorithm 3** The honest party

$\mathcal{G} \leftarrow \epsilon$
**function** CONSTRUCTOR($\mathcal{G}'$)
  $\mathcal{G} \leftarrow \mathcal{G}'$ ▷ Select Genesis Block
  $\mathcal{C} \leftarrow [\mathcal{G}]$ ▷ Add Genesis Block to start of chain
  round $\leftarrow 1$
**end function**
**function** EXECUTE($1^\kappa$)
  $\tilde{\mathcal{C}} \leftarrow \text{maxvalid}(C, \bar{M}[i])$ ▷ Adopt Longest Chain in the network
  **if** $\tilde{\mathcal{C}} \neq \mathcal{C}$ **then**
    BROADCAST($\tilde{\mathcal{C}}$) ▷ Gossip Protocol
  **end if**
  $x \leftarrow$ INPUT() ▷ Take all transactions in mempool
  $B \leftarrow \text{PoW}(x, \tilde{\mathcal{C}})$
  **if** $B \neq \perp$ **then** ▷ Successful Mining
    $\mathcal{C} \leftarrow \tilde{\mathcal{C}}||B$ ▷ Add block to current longest chain
    BROADCAST($\mathcal{C}$) ▷ Gossip protocol
  **end if**
  round $\leftarrow$ round+1
**end function**
**function** READ
  $x \leftarrow \epsilon$ ▷ Instantiate transactions
  **for** $B \in C$ **do**
    $x \leftarrow x||C.x$ ▷ Extract all transactions from each block in the chain
  **end for**
  **return** $x$
**end function**

## 4 Proof-of-Work

The algorithm below is run by miners to find a new block. Notice that all parties (adversarial and honest) have a maximum number of $q$ queries per round. This is to model the hash rate of parties. Furthermore, we construct a block as the concatenation of the previous block $s$, the transactions $x$, and the nonce $ctr$. For a block to be mined successfully, we require that $H(B) \leq T$, where $T$ is the mining target. Due to the size of the space $\{0,1\}^\kappa$, the probability of two parties mining with the same nonce is negligible, therefore we may assume that there are no "nonce collisions".

**Algorithm 4** The Proof-of-Work discovery algorithm

```
1: function POW_{H,T,q}(x, s)
2:     ctr ←$ {0,1}^κ                          ▷ Randomly sample Nonce
3:     for i ← 1 to q do                       ▷ Number of available queries per party
4:         B ← s||x||ctr                       ▷ Create block
5:         if H(B) ≤ T then                    ▷ Successful Mining
6:             return B
7:         end if
8:         ctr ← ctr +1
9:     end for
10:    return ⊥                                ▷ Unsuccessful Mining
11: end function
```

## 5 Longest Chain

This algorithm is run by honest nodes in order to adopt the longest chain each round. Since every honest node abides to the longest chain rule, the conditions are required for a chain to be adopted: the chain is valid and the chain is strictly longer (line 4). This algorithm is called in line 2 of the honest party algorithm: it will loop through every chain it received through the gossip network, check its validity and check that it is longer than the currently adopted chain.

**Algorithm 5** The maxvalid algorithm

```
1: function MAXVALID_{G,δ(·)}(C̄)
2:     C_max ← [G]                             ▷ Start with current adopted chain
3:     for C ∈ C̄ do                            ▷ Iterate for every chain received through gossip network
4:         if validate_{G,δ(·)}(C) ∧ |C| > |C_max| then   ▷ Longest Chain Rule
5:             C_max ← C
6:         end if
7:     end for
8:     return C_max
9: end function
```

## 6 Validating a block

This algorithm is used to validate a block, it is called in line 4 of the longest chain algorithm run by honest parties. The algorithm first checks that the Genesis block the chain is correct (line 2). Then for every block in the chain, the algorithm will update the UTXO, checking that each transaction is valid (lines 13-16). The algorithm will also check the PoW for each block and check that the block is in the correct format of $s||x||ctr$ (lines 9-12)

**Algorithm 6** The validate algorithm

```
1: function VALIDATE_{G,δ(·)}(C)
2:     if C[0] ≠ G then                          ▷ Check that first block is Genesis
3:         return false
4:     end if
5:     st ← st_0                                 ▷ Start at Genesis state
6:     h ← H(C[0])
7:     st ← δ*(st, C[0].x)
8:     for B ∈ C[1:] do                          ▷ Iterate for every block in the chain
9:         (s, x, ctr) ← B
10:        if H(B) > T ∨ s ≠ h then              ▷ PoW check and Ancestry check
11:            return false
12:        end if
13:        st ← δ*(st, B.x)          ▷ Application Layer: update UTXO & validate transactions
14:        if st = ⊥ then
15:            return false                      ▷ Invalid state transition
16:        end if
17:        h ← H(B)
18:    end for
19:    return true
20: end function
```

## 7 Chain Virtues

Equipped with this new rigorous definition of the environment, our assumptions and the algorithm ran by the honest party, we can now mathematically define the Chain Virtues, introduced earlier in the lectures.

1. **Common Prefix ($\kappa$).** $\forall$ honest parties $P_1, P_2$ adopting chains $C_1, C_2$ at any rounds $r_1 \leq r_2$ respectively, Common Prefix property $C_1[: -\kappa] \leq C_2$ holds.
2. **Chain Quality ($\mu, \ell$).** $\forall$ honest party $P$ with adopted chain $C$, $\forall i$ any chunk $C[i : i + \ell]$ of length $\ell > 0$ has a ratio of honest blocks $\mu$.
3. **Chain Growth ($\tau, s$).** $\forall$ honest parties $P$ and $\forall r_1, r_2$ with adopted chain $C_1$ at round $r_1$ and adopted chain $C_2$ at round $r_2 \geq r_1 + s$, it holds that $|C_2| \geq |C_1| + \tau s$.

We define a round during which one or more honest party found block as a **successful round ($r$).** A round has a **convergence opportunity($r$)** if only one honest party found a block irrespective of adversarial parties.

## 8 Pairing Lemma

**Lemma 8.1.** *Let $B = C[i]$ for some chain $C$ s.t. $B$ was computed by an honest party $P$ during a convergence opportunity $r$. Then for any block $B'$ at position $i$ of some other chain $C'$, if $B \neq B'$, then $B'$ was adversarially computed.*

*Proof.* Suppose for contradiction that $B'$ was mined on a round $r'$. For the sake of contradiction, assume that $B'$ was honestly computed. Thus, we need to analyze three following cases:

1. Case 1: $r = r'$. This is not possible as round $r$ was a convergence opportunity.
2. Case 2: $r < r'$. This is not possible as due to the longest chain rule, after round $r$, everybody will have adopted a chain of at least $i$ blocks, so honest parties would not accept $B'$.
3. Case 3: $r > r'$. This is not possible, same as above but this time honest parties wouldn't adopt block $B$.

So we have a contradiction, thus, $B'$ must have been adversarially mined. □

From the above, we note that, if the adversary wants to displace block $B$, she has to pay for it by mining $B'$. Therefore, if the adversary does not mine a block, than the convergence opportunity will be a true honest convergence.

## 9 Honest Majority Assumption $(n, t, \delta)$

We will now give a new definition of the honest majority assumption by introducing the honest advantage parameter $\delta$ . We will see in the next lecture that we need this parameter in order the chain virtues hold for Bitcoin. We say that the honest majority assumption holds if $t < (1-\delta)(n-t)$.

## References


[1] N. L. Juan A. Garay, Aggelos Kiayias. The Bitcoin Backbone Protocol: Analysis and Applications, 2020. `https://eprint.iacr.org/2014/765.pdf`.

[2] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, 2008. `https://bitcoin.org/bitcoin.pdf`.