

# IACR Policy for Cryptology Schools

August 2015\*

## 1 Related Works

Nodes on [4]:

- This model starts with describing how to model execution of *synchronous* protocols that can access a global setup clock.
- In a previous treatment, the clock in UC was local to each party and it would have to receive update messages from the other parties (everyone is doing this operation). Hence, with GUC the environment can control the clock speed and define when clock updates happen (as other protocol sessions might also be accessing it).

There are several works from the past few years that try to model a blockchain within the Universal Composability framework—some attempting to model it in its extension, (G)UC [?, ?].

Kiayias et al. [4] models a Bitcoin-like blockchain for fair and robust multi-party computation. It is motivated by the impossibility result for fairness in secure MPC<sup>1</sup> and circumventing it by imposing monetary penalties on participants. The model consists of two global functionalities,  $\overline{\mathcal{G}}_{\text{clock}}$  and  $\overline{\mathcal{G}}_{\text{blockchain}}$ . The blockchain functionality enables the expected functionality like submitting transactions, validating them, batching them into blocks, and allowing an adversary to reorder transactions. Because of the GUC framework, the state of the blockchain is available to all parties including the environment and any other protocol sessions (or dummy parties). This work however, fails to prove that their model of the blockchain is GUC-realized in any currently existing blockchain system. Such a security proof is essential as it provides credibility to the possibility of implementing protocols in the  $\overline{\mathcal{G}}_{\text{blockchain}}$ -hybrid world. Furthermore, the assumptions that are made for the blockchain and what the adversary can do severely limit the scope of adversaries in the real-world. The first failure of this model is to consider an adversary which can change the view some parties have of the blockchain state. For example, if the adversary mines a new block and keeps it a secret, or if some nodes have not received new blocks because of communication delays. Another failure is that all transactions in the buffer between blocks are always included in the next block. This, again, prevents a miner-like adversary which can censor transactions and delay their entry into the chain. Finally, the state of the blockchain

---

\*The most recent version of this document can be obtained from <http://www.iacr.org/docs/>.  
Editors of this document: M. Abdalla, A. Boldyreva, C. Cachin, A. Kiayias, B. Warinschi (2014).

<sup>1</sup>Fairness in MPC is defined as: either all parties learn the output or none of them do.

is updated at fixed time intervals which does not accurately convey the consensus model of Bitcoin or Ethereum.

Badertscher et al. [1] attempt to solve these problems by allowing a more unrestricted in the GUC framework. The shared functionality in this case is a global clock functionality,  $\overline{\mathcal{G}}_{\text{clock}}$ , which enables modelling a synchronous system in the UC framework by proceeding in rounds. Because it is a shared functionality, the clock allows any other protocol session in the environment to be synchronized with the challenge protocol. The blockchain functionality is a local functionality (only available to the parties within the protocol session) that allows the adversary to have more power in what it can do. The adversary can inject transactions and modify the state of the chain that all parties that query it can see. This is accomplished by allowing a maximum distance,  $d$ , that the adversary can specify and return a prefix of the chain which is at most a distance  $d$  from the head of the chain. Furthermore, the adversary can choose exactly which transactions are allowed to be in the next block. The blockchain functionality is modularized by allowing the definition of subroutines that capture extending the blockchain state (specifically for Bitcoin in this paper). The authors of this work admit that the paper's only intent is to model the Bitcoin blockchain hence the choice to use the ledger as only a local functionality. This prevents other protocol sessions from using the same blockchain (definitely a limitation of modelling the reality of a blockchain environment). Furthermore, this paper makes the argument that it is dangerous to have a global ledger functionality as such replacement does not “in general, preserve a realization proof of some ideal functionality  $\mathcal{F}$  that is conducted in a ledger-hybrid world, because the simulator in that proof might rely on specific capabilities that are not available any more after the replacement (as the global setup is also replaced in the real world)”. It claims that [2] provides a sufficient condition for such a replacement, but that the condition is too strong to be satisfied by any ledger implementation.

Canneti et al. [3] addresses the global PKI and an ideal authentication within the UC with global setup. The specific problem presented in this paper is that the ideal authentication functionality,  $\mathcal{F}_{\text{auth}}$ , is usually formulated with the desirable property of non-transferrability of authentication. This means that when I send an authenticated message to another person, they are unable to use that proof to convince anyone else of the authentication. The paper realized that the real world PKI model is global *and* that, within it, signatures are globally verifiable. Once a key has signed a message for authentication, that proof is verifiable by and transferrable to anyone else in the system. Therefore, this work models a new relaxed global PKI, relaxes the UC authentication protocol to not require deniability, and formulates new functionalities for authentication and key exchange without deniability. Finally, they propose a new composition theorem allowing substitution of global functionalities,  $\mathcal{F}$  *EUC-realizes*  $\mathcal{G}$ . The problem being solved relates back to a claim made by Badertscher et al. [1] that replacement of global functionalities with real implementations generally invalidates a realization proof of some functionality that shares state with it. In this paper, this arises as replacement of the UC PKI system with a real one where transferrability is possible invalidates the realization proof of the ideal authentication functionality in the plain-PKI model.

They formulate a new authentication functionality that does not impose non-transferrability and a long lasting global functionality handling certificates. Finally they prove that the certificate functionality guarantees are precisely captured by EU-CMA signatures and a globally-available PKI. This paper however imposes some restrictions on what can be done. For example, there is a limitation that a particular ITI may only register a single key with the Cert and Bulletin Board functionalities. They claim however, that it is possible to realize  $\mathcal{F}_{\text{cert\_auth}}$ , but a

certificate-based approach is not it.

One of the main takeaways in this paper is that you can define a functionality and analyze it for its properties then prove that it is equivalent to another functionality that realizes this protocol. In this paper that is done by defining

**Differentiating  $\mathcal{G}_{\text{cert}}^{\text{pid}}$  and  $\mathcal{G}_{\text{swk}}^{\text{pid}}$ .** Questions to answer:

- What is the precise difference between  $\mathcal{G}_{\text{cert}}^{\text{pid}}$  and  $\mathcal{G}_{\text{swk}}^{\text{pid}}$  and why is the substitution necessary?

**ExecTx(to, val, data, from)**

```

nonces[from]  $\leftarrow$  nonces[from] + 1
If balances[from] < val: reject
balances[from]  $\leftarrow$  balances[from] - val
balances[to]  $\leftarrow$  balances[to] + val
receipts[from, nonces[from]]  $\leftarrow$  CreateTxRef(val, from)
If to  $\in$  contracts:
  ret  $\leftarrow$  Exec(to, val, data, from)
  txs[from, nonces[from]]  $\leftarrow$  ret

```

**ExecContractCreate(addr, val, data, from, private)**

```

nonces[from]  $\leftarrow$  nonces[from] + 1
If balances[from] < val: reject
balances[from]  $\leftarrow$  balances[from] - val
balances[to]  $\leftarrow$  balances[to] + val
(functions, args) := data
r  $\leftarrow$  functions.init(args)
contracts[addr] = functions
restricted[addr] = private
If  $\neg r$ :
  balances[from]  $\leftarrow$  balances[from] + val
  balances[to]  $\leftarrow$  balances[to] - val

```

## References

- [1] Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Bitcoin as a transaction ledger: A composable treatment. In *Annual International Cryptology Conference*, pages 324–356. Springer, 2017.
- [2] Ran Canetti, Daniel Shahaf, and Margarita Vald. Universally composable authentication and key-exchange with global pki. In *IACR International Workshop on Public Key Cryptography*, pages 265–296. Springer, 2016.

$\bar{\mathcal{G}}_{\text{ledger}}$ 

Initialize  $\text{txqueue} := \{\}, \text{contracts} := \{\}, \text{newtxs} := \{\}, \text{nonces} := \{\}, \text{balances} := \{\},$   
 $\Delta := 8, \text{rnd} := 0$

**On input** (transfer, to, val, data, from) from  $\mathbf{P}_i = (\text{sid}, \text{pid})$ :

If  $\text{balances}[\text{fro}] < \text{val}$ : **reject**

$\text{nonces}[\text{from}] \leftarrow \text{nonces}[\text{from}] + 1$

$\text{newtxs}[\text{from}, \text{nonces}[\text{from}]] \leftarrow (\text{transfer}, \text{to}, \text{val}, \text{data}, \text{from})$

**leak** (transfer, to, val, data, from) to  $\mathcal{A}$

**On input** (contract create, addr, val, data, private, from) from  $\mathbf{P}_i = (\text{sid}, \text{pid})$ :

If  $\text{balances}[\text{from}] < \text{val}$ : **reject**

$\text{nonces}[\text{from}] \leftarrow \text{nonces}[\text{from}] + 1$

$\text{caddr} \leftarrow \text{ComputeAddr}(\text{from})$

If  $\text{caddr} \neq \text{addr}$ : **reject**

If  $\text{len}(\text{data}) = 0$ : **reject**

$\text{newtxs}[\text{from}, \text{nonces}[\text{from}]] \leftarrow (\text{transfer}, \text{to}, \text{val}, \text{data}, \text{from})$

**leak** (contract create, addr, val, data, private, from) to  $\mathcal{A}$

**On input** (tick, addr) from  $\mathbf{P}_i = (\text{sid}, \text{pid})$ :

$\text{rnd} += 1$

$\text{balances}[\text{addr}] += 1000000$

**For** tx **in**  $\text{txqueue}[\text{rnd}]$ :

**If**  $\text{tx}[0] = \text{transfer}$ :

$(\text{transfer}, \text{to}, \text{val}, \text{data}, \text{from}) \leftarrow \text{tx}$

$\text{ExecTx}(\text{to}, \text{val}, \text{data}, \text{from})$

**If**  $\text{tx}[0] = \text{contractcreate}$ :

$(\text{contractcreate}, \text{addr}, \text{val}, \text{data}, \text{private}, \text{from}) \leftarrow \text{tx}$

$\text{ExecContractCreate}(\text{addr}, \text{val}, \text{data}, \text{private}, \text{from})$

---

**On input** (delayTx, from, nonce, rounds) from  $\mathcal{A}$ :

$\text{tx} \leftarrow \text{newtxs}[\text{from}, \text{nonce}]$

  Add tx to  $\text{txqueue}[\text{rnd} + \text{rounds}]$

  Remove tx from newtxs

**On input** (tick, addr, permutation) from  $\mathcal{A}$ :

  Apply *permutation* to  $\text{txqueue}[\text{rnd}]$

  Run honest party mining with addr

Figure 1: Ideal functionality representing a basic ledger with adversarial methods for delaying/reordering transactions and smart contract support

- [3] Ran Canetti, Daniel Shahaf, and Margarita Vald. Universally composable authentication and key-exchange with global pki. In *IACR International Workshop on Public Key Cryptography*, pages 265–296. Springer, 2016.
- [4] Aggelos Kiayias, Hong-Sheng Zhou, and Vassilis Zikas. Fair and robust multi-party compu-

**Protection Wrapper  $\mathcal{W}_p$** 

**On input** (transfer, to, val, data, from) from  $\mathbf{P}_i = (\text{sid}, \text{pid})$ :  
 $to \leftarrow$

Figure 2: Protection wrapper for the ledger to maintain indistinguishability.

 **$\Pi_{\text{pay}}$ : Contract<sub>pay</sub>**

**Init** ( $P_L, P_R$ ):  
 $\text{deposits}_L, \text{deposits}_R := 0$   
**On input** (deposit) (tx) from  $\mathbf{P}_i = (\text{sid}, \text{pid})$ :  
 $\text{deposits}_i += \text{tx.value}$   
 $\text{out}(\text{deposits}_L, \text{deposits}_R)$   
**On input** (output, aux<sub>out</sub>, tx) :  
 parse aux<sub>out</sub> as ( $\text{wd}_L, \text{wd}_R$ )  
**For**  $i \in \{L, R\}$ :  $\text{send}(P_i, \text{wd}_i)$

Figure 3: Contract pay

 **$\Pi_{\text{pay}}$** 

Initialize  $\text{arr}_i = \emptyset, \text{pay}_i = 0, \text{wd}_i = 0, \text{paid}_i = 0$   
 Contract<sub>pay</sub> identifier  $\mathcal{C}$   
**Send**  $(\emptyset, 0) \rightarrow \mathcal{F}_{\text{state}}$   
**On input** ( $\text{cred}_L, \text{new}_L, \text{cred}_R, \text{new}_R$ ) from  $\mathcal{F}_{\text{state}}$ :  
**For**  $e \in \text{new}_i$ :  
**Output** (receive,  $e$ )  
 $\text{paid}_i += e$   
**Send**  $(\text{arr}_i, \text{wd} - \text{wd}_n) \rightarrow \mathcal{F}_{\text{state}}$   
 $\text{arr}_i \leftarrow \emptyset$   
 $\text{wd}_n \leftarrow \text{wd}_i$   
**On input** (pay,  $\$X$ ) from  $\mathcal{Z}$ :  
 $\text{Contract}_{\text{pay}} \leftarrow \mathcal{G}_{\text{ledger}}.\text{contract}(\mathcal{C})$   
**If**  $\$X \leq \text{Contract}_{\text{pay}}.\text{deposits}_i + \text{paid}_i - \text{pay}_i - \text{wd}_i$ :  
 $\text{arr}_i \leftarrow \$X$   
 $\text{pay}_i += \$X$   
**On input** (withdraw,  $\$X$ ) from  $\mathcal{Z}$ :  
 $\text{Contract}_{\text{pay}} \leftarrow \mathcal{G}_{\text{ledger}}.\text{contract}(\mathcal{C})$   
**If**  $\$X \leq \text{Contract}_{\text{pay}}.\text{deposits}_i + \text{paid}_i - \text{pay}_i - \text{wd}_i$ :  
 $\text{wd}_i += \$X$

Figure 4: Local prot pay

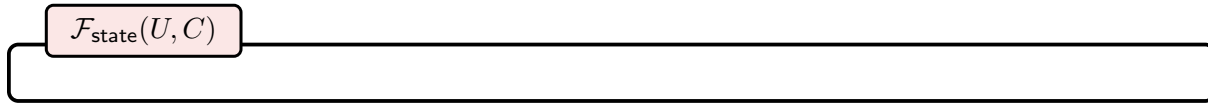


Figure 5: fstate

tation using a global transaction ledger. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 705–734. Springer, 2016.