Verifying liquidity of Bitcoin contracts

(oral communication)

Massimo Bartoletti
Università di Cagliari

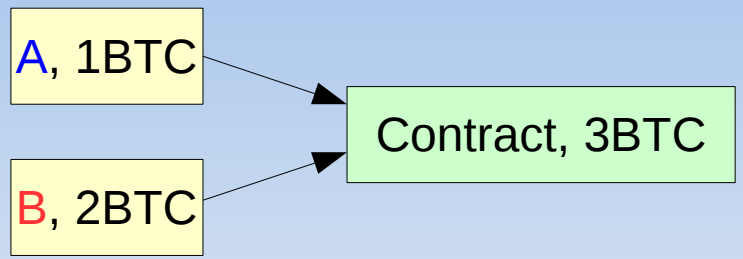
Roberto Zunino
Università di Trento

DLT 2019, Pisa, 2019-02-12

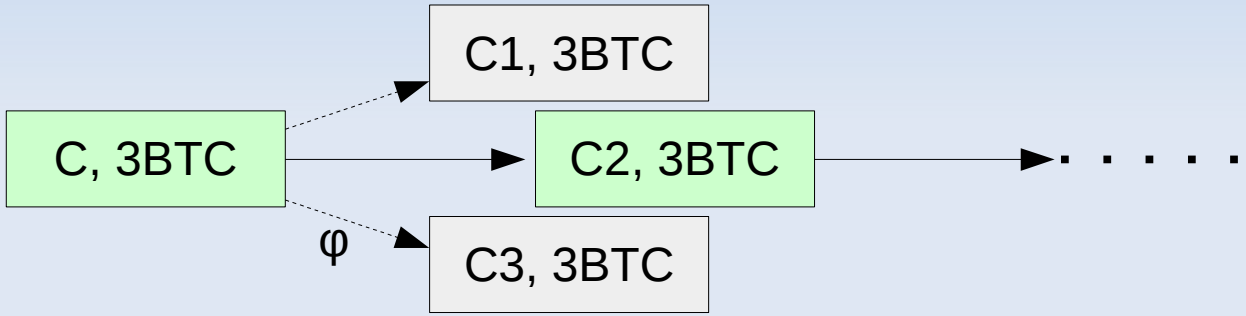
The life of smart contracts, simply put

Smart contracts: basic workflow

Stipulation

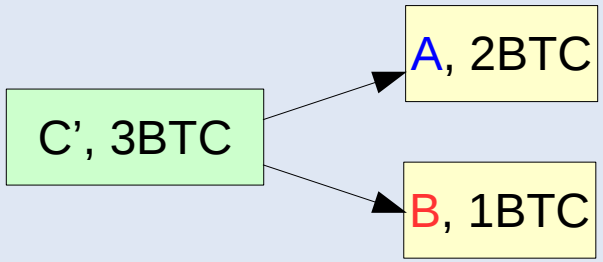


Execution
(state updates)

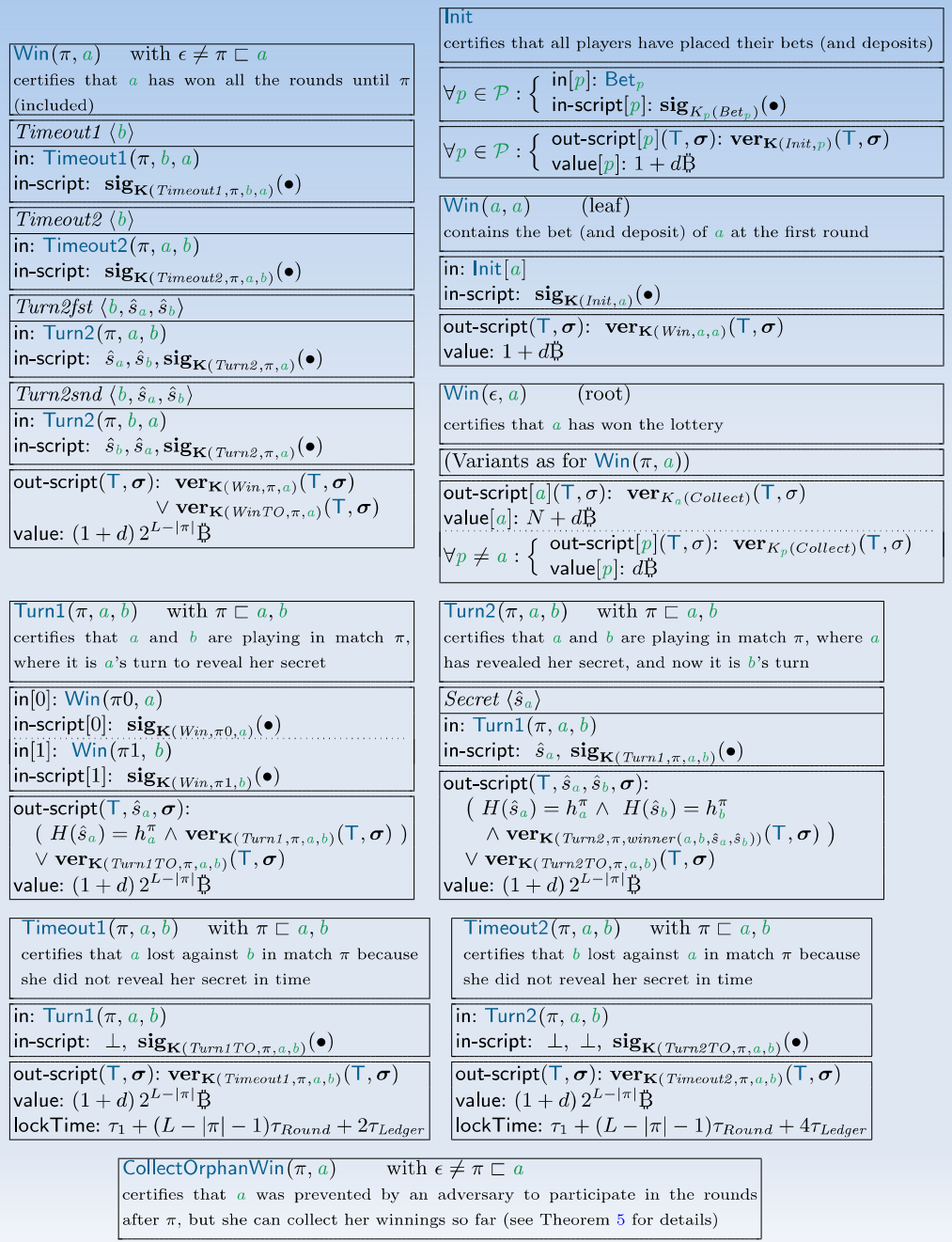


contracts describe possible moves (transition system)
participants choose which moves to perform (strategy)

Termination



Designing low-level smart contracts is hard!



A lottery smart contract on Bitcoin

[BITCOIN 2017]

Very error-prone to design!
Hard to guarantee security

We need high level languages:

EVM → Solidity
Bitcoin → BitML

BitML in a nutshell

- A high-level language for smart contracts on Bitcoin

[ACM CCS 2018]

- Main features:
  - Depositing / withdrawing cryptocurrency
  - Committing to secrets (& revealing them)
  - Time constraints
  - Authorization-enabled actions
- Not Turing-complete, but can model timed commitment, escrow contracts, micropayment channels, lotteries, ...

BitML Contract Example

$$
\begin{aligned}
&\mathsf{init}\,\{A : 1\,\text{฿}, \mathsf{secret}\,a \\
&\qquad B : 1\,\text{฿}, \mathsf{secret}\,b\} \\
&\\
&(\mathsf{reveal}\,a. \\
&\qquad (\ \mathsf{reveal}\,b.\ \mathsf{if}(a+b)\%2 = 0 \\
&\qquad\qquad\qquad \mathsf{then\ withdraw}\ A \\
&\qquad\qquad\qquad \mathsf{else\ withdraw}\ B \\
&\qquad +\mathsf{after}\,2\cdot \mathrm{t} :\ \ \mathsf{withdraw}\ A) \\
&+\mathsf{after}\,\mathrm{t} :\ \ \mathsf{withdraw}\ B)
\end{aligned}
$$

BitML security

- Computationally sound compilation to Bitcoin

  no BitML attacks $\Rightarrow$ no Bitcoin attacks

- To guarantee Bitcoin-level security, we still need to verify BitML code against desirable properties
- Liquidity is a desirable general property of smart contracts

Liquidity

- Let S be a strategy for a participant interacting with a given contract C
- Intuition:

  S is liquid for C iff, even in the presence of adversaries, S can eventually cause the contract balance to be assigned to participants (in some way)

$$\text{reveal}\, a.\ \text{reveal}\, b.\ \text{split}(1\text{฿} \to \text{withdraw A} \mid 1\text{฿} \to \text{withdraw B})$$

no liquid strategy for A

$$\text{reveal}\, a.\ (\ \text{reveal}\, b.\ \text{split}(1\text{฿} \to \text{withdraw A} \mid 1\text{฿} \to \text{withdraw B}) + \text{after}\ t\ .\ \text{withdraw A})$$

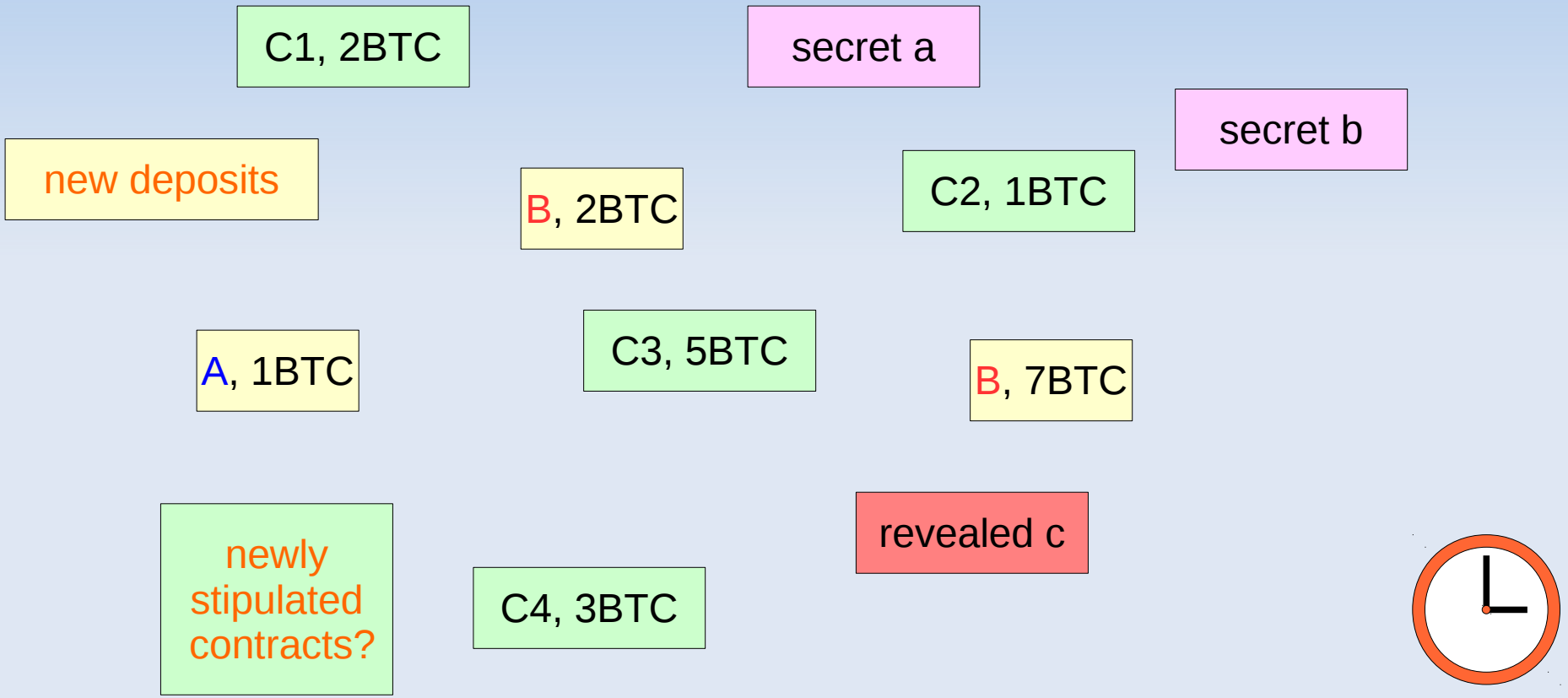
liquid strategy for A: reveal and wait

- Ethereum Parity attack violated liquidity

Liquidity variants

- Basic: from any reachable state of C, strategy S can perform a sequence of moves “liquidating” C
- Multiparty: a set of participants cooperate to make C terminate
- Quantitative: we don’t need C to terminate, as long as a large enough part of its balance is distributed
- Known/unknown secrets: S should be able to “liquidate” C no matter what the adversary secrets are

BitML Abstraction



The BitML transition system is infinite-state, infinite branching, and timed

BitML Abstraction

Focus on a given contract, only, and forget the irrelevant part of the configuration

C1, 2BTC

secret a

secret b

new deposits

B, 2BTC

C2, 1BTC

A, 1BTC

C3, 5BTC

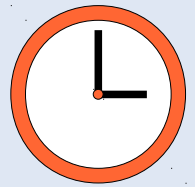
B, 7BTC

revealed c

newly stipulated contracts?

C4, 3BTC

Abstract time exploiting C3



The abstract transition system is now finite-state!

Main Result

- Our abstraction is sound and complete w.r.t. liquidity

[to appear in POST 2019]

- Corollary: liquidity in BitML is decidable

- Verification tool in development (by UniCA)

Further directions

- Strategy inference
  - Given a contract, find a strategy for a participant maximizing their payoff
- Probabilistic analysis
  - E.g. what is the average payoff?
  - Useful for lotteries

Thank you

(all papers available on IACR)