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

A, 1BTC → Contract, 3BTC
B, 2BTC

Execution (state updates)

C, 3BTC → C1, 3BTC
φ
C3, 3BTC → C2, 3BTC...

Termination

C’, 3BTC → A, 2BTC
B, 1BTC

contracts describe possible moves (transition system) participants choose which moves to perform (strategy)
Designing low-level smart contracts is hard!

We need high level languages:

EVM → Solidity
Bitcoin → BitML

A lottery smart contract on Bitcoin

[BITCOIN 2017]

Very error-prone to design! Hard to guarantee security

Win(\pi, a) \text{ with } \epsilon \neq \pi \sqsubseteq a
\text{certifies that } a \text{ has won all the rounds until } \pi \text{ (included)}

Timeout(\pi)
\text{in: } Timeout(\pi, b, a)
\text{in-script: } \text{sig}_{K_{(\text{Timeout}, \sigma, a), \epsilon}}(\bullet)

Timeout2(\pi)
\text{in: } Timeout2(\pi, a, b)
\text{in-script: } \text{sig}_{K_{(\text{Timeout2}, \sigma, a), \epsilon}}(\bullet)

TurnSilk(\delta, \delta, \delta)
\text{in: } Turn(\pi, \delta, a, b)
\text{in-script: } \delta, \delta, \delta, \text{sig}_{K_{(\text{Turn}, \sigma, a), \epsilon}}(\bullet)

TurnSend(\delta, \delta, \delta)
\text{in: } Turn2(\pi, \delta, a, b)
\text{in-script: } \delta, \delta, \delta, \delta, \text{sig}_{K_{(\text{Turn2}, \sigma, a), \epsilon}}(\bullet)

\text{out-script}(T, \sigma): \text{ver}_{K_{(\text{Win}, \pi, a), \epsilon}}(T, \sigma)
\text{value: } (1 + d)2^{L-\pi-\delta}

Turn(\pi, a, b) \text{ with } \pi \sqsubseteq a, b
\text{certifies that } a \text{ and } b \text{ are playing in match } \pi,
\text{where } a \text{ is } \epsilon \text{’s turn to reveal her secret}

in[0]: \text{Win}(\pi, 0, a)
\text{in-script[0]: } \text{sig}_{K_{(\text{Win}, \pi, 0, a)}}(\bullet)
\text{in[1]: } \text{Win}(\pi, 1, b)
\text{in-script[1]: } \text{sig}_{K_{(\text{Win}, \pi, 1, b)}}(\bullet)

\text{out-script}(T, \delta, \sigma):
( H(\delta) = h^\pi \wedge \text{ver}_{K_{(\text{Turn}, \pi, a, b)}}(T, \sigma) )
\wedge \text{ver}_{K_{(\text{Turn2}, \pi, a, b)}}(T, \sigma)
\text{value: } (1 + d)2^{L-\pi-\delta}

Timeout(\pi, a, b) \text{ with } \pi \sqsubseteq a, b
\text{certifies that } a \text{ lost against } b \text{ in match } \pi \text{ because she did not reveal her secret in time}

\text{in: } Turn(\pi, a, b)
\text{in-script: } a, a, \text{sig}_{K_{(\text{Turn}, \pi, a, b)}}(\bullet)

\text{out-script}(T, \sigma): \text{ver}_{K_{(\text{Timeout}, \pi, a, b)}}(T, \sigma)
\text{value: } (1 + d)2^{L-\pi-\delta}
\text{lockTime: } t_1 + (L - |\pi| - 1)T_{\text{Round}} + 2T_{\text{Ledger}}

Timeout2(\pi, a, b) \text{ with } \pi \sqsubseteq a, b
\text{certifies that } b \text{ lost against } a \text{ in match } \pi \text{ because she did not reveal her secret in time}

\text{in: } Turn2(\pi, a, b)
\text{in-script: } a, a, a, \text{sig}_{K_{(\text{Turn2}, \pi, a, b)}}(\bullet)

\text{out-script}(T, \sigma): \text{ver}_{K_{(\text{Timeout2}, \pi, a, b)}}(T, \sigma)
\text{value: } (1 + d)2^{L-\pi-\delta}
\text{lockTime: } t_1 + (L - |\pi| - 1)T_{\text{Round}} + 4T_{\text{Ledger}}

CollectOrphanWin(\pi, a) \text{ with } \epsilon \neq \pi \sqsubseteq a
\text{certifies that } a \text{ was prevented by an adversary to participate in the rounds after } \pi, \text{but she can collect her winnings so far (see Theorem 3 for details)}
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, ...
init \( \{ \text{\textbf{A}} : 1 \text{ B}, \text{secret } a \}
\begin{align*}
\text{\textbf{B}} : 1 \text{ B}, \text{secret } b \}\}
\end{align*}

(reveal \( a \).
\begin{align*}
(\text{reveal } b. \text{ if } (a + b)^\%2 = 0 \\
\text{then withdraw } \text{\textbf{A}} \\
\text{else withdraw } \text{\textbf{B}} \\
+ \text{after } 2 \cdot t : \text{ withdraw } \text{\textbf{A}}) \\
+ \text{after } t : \text{ withdraw } \text{\textbf{B}})
\end{align*}
BitML security

- Computationally sound compilation to Bitcoin
  
  no BitML attacks \implies 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).

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

  no liquid strategy for $A$

reveal $a$.

  ( reveal $b$. split($1B \rightarrow \text{withdraw}A \mid 1B \rightarrow \text{withdraw}B$)  
  
  + after $t$. 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
The BitML transition system is infinite-state, infinite branching, and timed
Focus on a given contract, only, and forget the irrelevant part of the configuration.

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)