Developing secure Bitcoin contracts with BitML

Stefano Lande
Nicola Atzei
Massimo Bartoletti
Nobuko Yoshida
Roberto Zunino

University of Cagliari
Imperial College London
University of Trento
Contracts as programs vs contracts as protocols

<table>
<thead>
<tr>
<th></th>
<th>Contracts as programs</th>
<th>Contracts as protocols</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complexity of blockchain design</td>
<td>High (gas, VM, compilers,…)</td>
<td>Low (well understood security)</td>
</tr>
<tr>
<td>Ease of programming</td>
<td>High (Solidity, …)</td>
<td>Low (Protocols + redeem scripts)</td>
</tr>
<tr>
<td>Automatic verification</td>
<td>Yes (sound ⇒ not complete)</td>
<td>No (1 contract → 1 proof)</td>
</tr>
</tbody>
</table>

Can we get the best of both?  
(without creating a new coin)
### Smart contracts on Bitcoin

<table>
<thead>
<tr>
<th><strong>Commit</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>out</strong>: 1 BTC:</td>
</tr>
<tr>
<td>fun ( x \sigma ). ( ( H(x) = h ) and ( \text{ver}_A(\sigma) ) )</td>
</tr>
<tr>
<td>or <strong>afterAbs t</strong>: ( \text{ver}_B(\sigma) )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Timeout</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>wit</strong>: * ( \text{sig}_B(\text{Timeout}) )</td>
</tr>
<tr>
<td><strong>absLock</strong>: t</td>
</tr>
</tbody>
</table>

**Pre-condition:**

1. The key pair of \( C \) is \( \bar{C} \) and the key pair of each \( P_i \) is \( \bar{P}_i \).
2. The Ledger contains \( n \) unredeemed transactions \( U^C_1, \ldots, U^C_n \), which can be redeemed with key \( \bar{C} \), each having value \( dB \).
3. **The CS.Commit(\( C, d, t, s \)) phase**
   - The Committer \( C \) computes \( h = H(s) \). He sends to the Ledger the transactions \( \text{Commit}_1, \ldots, \text{Commit}_n \). This obviously means that he reveals \( h \), as it is a part of each \( \text{Commit}_i \).
   - If within time \( \text{maxLedge} \), some of the \( \text{Commit}_i \) transactions does not appear on the Ledger, or if they look incorrect (e.g. they differ in the \( h \) value) then the parties abort.
4. The Committer \( C \) creates the bodies of the transactions \( \text{PayDeposit}_1, \ldots, \text{PayDeposit}_n \), signs them and for all \( i \) sends the signed body \( \text{PayDeposit}_i \) to \( P_i \). If an appropriate transaction does not arrive to \( P_i \), then he halts.
5. **The CS.Open(\( C, d, t, s \)) phase**
6. The Committer \( C \) sends to the Ledger the transactions \( \text{Open}_1, \ldots, \text{Open}_n \), what reveals the secret \( s \).
7. If within time \( t \) the transaction \( \text{Open}_i \) does not appear on the Ledger then \( P_i \) signs and sends the transaction \( \text{PayDeposit}_i \) to the Ledger and earns \( dB \).

Languages for Bitcoin scripts

**Balzac** (UniCa)

```plaintext
transaction T_commit(h, deadline) {
  input = A_funds: sig(kA)
  output = this.input.value:
    fun(x,s:string) .
      sha256(s) == h && versig(kApub;x)
      || checkDate deadline: versig(kBpub;x)
}
```

**Miniscript** (Blockstream)

```plaintext
or(
  and(pk(A), sha256(H)),
  and(pk(B), after(deadline))
)`
BitML: Bitcoin Modelling Language

\[ C ::= D_1 + \cdots + D_n \]  
contract

\[ D ::= \]

 withdrawal A 

 transfer balance to A

 split \( v_1 \rightarrow C_1 | \cdots | v_n \rightarrow C_n \) 

 split balance

 A : D 

 wait for A’s authorization

 after t : D 

 wait until time t

 put x . C 

 collect deposit x

 reveal a b \ldots \text{ if } p . C 

 reveal secrets a, b, ...
A basic example

Precondition: A must put a 1฿:

\{A : !1฿\}

Contract:

\text{PayOrRefund} = A:\text{withdraw} B + B:\text{withdraw} A

Problem: if neither A nor B give their authorization, the 1฿ deposit is frozen
Mediating disputes (with oracles)

Resolve disputes via a mediator $M$ (paid $0.2\mathregular{฿}$)

\[
\text{Escrow} = A:\text{withdraw } B + B:\text{withdraw } A \\
+ A:\text{Resolve} + B:\text{Resolve}
\]

\[
\text{Resolve} = \text{split} \\
0.2\mathregular{฿} \to \text{withdraw } M \\
| 0.8\mathregular{฿} \to M:\text{withdraw } A + M:\text{withdraw } B
\]
The timed commitment in BitML

Precondition:

\{A: !1B \mid A: \text{secret } a\}

Contract:

reveal a. withdraw A

+ after t : withdraw B
The compiled timed commitment

<table>
<thead>
<tr>
<th>Tinit</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>out</strong>: 1 BTC:</td>
</tr>
<tr>
<td>function ( x \sigma \sigma' ) ( H(x) = h ) and ( ver_{A,B}(\sigma, \sigma') ),</td>
</tr>
<tr>
<td>or ( \text{afterAbs } t: \ ver_{A,B}(\sigma, \sigma') )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>wit</strong>: ( s \ sig_A \ sig_B )</td>
</tr>
<tr>
<td><strong>out</strong>: 1 BTC:</td>
</tr>
<tr>
<td>function ( \sigma ), ( ver_A(\sigma) )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TB</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>wit</strong>: ( 0 \ sig_A \ sig_B )</td>
</tr>
<tr>
<td><strong>out</strong>: 1 BTC:</td>
</tr>
<tr>
<td>function ( \sigma ), ( ver_B(\sigma) )</td>
</tr>
<tr>
<td><strong>absLock</strong>: ( t )</td>
</tr>
</tbody>
</table>
A 2-players lottery

\{A:!3฿ | A:secret a | B:!3฿ | B:secret b\}

split

2฿ → reveal b . withdraw B
+ after t : withdraw A

|2฿ → reveal a . withdraw A
+ after t : withdraw B

|2฿ → reveal a b if a=b . withdraw A
+ reveal a b if a≠b . withdraw B
A 2-players lottery (fair version)

\[
\{A:!3฿ | A:\text{secret} \ a | B:!3฿ | B:\text{secret} \ b\}
\]

split

\[2฿ \to \text{reveal} \ b \ \text{if} \ 0 \leq b \leq 1 . \ \text{withdraw} \ B\]
\[+ \ \text{after} \ t : \ \text{withdraw} \ A\]

\[|2฿ \to \text{reveal} \ a . \ \text{withdraw} \ A\]
\[+ \ \text{after} \ t : \ \text{withdraw} \ B\]

\[|2฿ \to \text{reveal} \ a \ b \ \text{if} \ a=b . \ \text{withdraw} \ A\]
\[+ \ \text{reveal} \ a \ b \ \text{if} \ a \neq b . \ \text{withdraw} \ B\]
Compiler security

BitML

Bitcoin
Verification

BitML supports the automatic verification of contract properties.

- Contract-dependent properties (expressed as LTL formulae)
  
  \[
  \text{TimedCommitment} \models \Box \Diamond (B \text{ knows } a \text{ or } B \text{ has } 1 \text{ BTC})
  \]

- Liquidity: funds are never “frozen” in the contract (⇒ Eth Parity Wallet)

  \[A:B:withdraw \ C \ + \ A:B:withdraw \ D\]

No liquid strategy for A, because A requires the cooperation of B
BitML toolchain

https://github.com/bitml-lang/
Benchmarks & tool demo

<table>
<thead>
<tr>
<th>Contract</th>
<th># Part</th>
<th># Tx</th>
<th>Ver. time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mutual timed commitment</td>
<td>2</td>
<td>15</td>
<td>83 ms</td>
</tr>
<tr>
<td>Escrow</td>
<td>3</td>
<td>12</td>
<td>8 s</td>
</tr>
<tr>
<td>Coupon Bond</td>
<td>3</td>
<td>18</td>
<td>1.3 s</td>
</tr>
<tr>
<td>Lottery</td>
<td>2</td>
<td>8</td>
<td>142 ms</td>
</tr>
<tr>
<td>Lottery</td>
<td>4</td>
<td>587</td>
<td>67 h</td>
</tr>
<tr>
<td>Rock Paper Scissors</td>
<td>2</td>
<td>23</td>
<td>781 ms</td>
</tr>
<tr>
<td>Morra</td>
<td>2</td>
<td>40</td>
<td>674 ms</td>
</tr>
<tr>
<td>Auction</td>
<td>2</td>
<td>42</td>
<td>3 s</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

All the tx are standard (→ they respect the 520-bytes constraint)
BitML wishlist #1

**Bitcoin completeness**: extend BitML to make it expressive as Bitcoin

- Find participants at runtime
- SIGHASH modes
- Relative timelocks
- Dynamic stipulation of subcontracts

Add 1 BTC. C

In 3 days: C

A: B: new C(x)
BitML wishlist #2

Currently, each step in the execution of a contract corresponds to an on-chain transaction

**BitML layer 2:**
- Execute BitML contracts off-chain
- In case of dispute, revert to on-chain execution
BitML wishlist #3

BitML over Taproot

- Exploit forthcoming MAST and Schnorr signatures
- Unexecuted script branches remain off-chain
  - More space efficient
  - Increases expressivity (520 bytes limit)
- Private: hides unexecuted script branches
References

N. Atzei, M. Bartoletti, S. Lande, N. Yoshida, R. Zunino
Developing secure Bitcoin contracts with BitML. ESEC/FSE, 2019

M. Bartoletti, R. Zunino.
BitML: a calculus for Bitcoin smart contracts. ACM CCS, 2018

M. Bartoletti, R. Zunino
Verifying liquidity of Bitcoin contracts. POST 2019

M. Bartoletti, T. Cimoli, R. Zunino.
Fun with Bitcoin smart contracts. ISOLA 2018

N. Atzei, M. Bartoletti, T. Cimoli, S. Lande, R. Zunino.
SoK: unraveling Bitcoin smart contracts. POST 2018

N. Atzei, M. Bartoletti, S. Lande, R. Zunino.
A formal model of Bitcoin transactions. Financial Cryptography, 2018
The BitML toolchain

- Tutorial: [https://blockchain.unica.it/bitml](https://blockchain.unica.it/bitml)
- Demo: [https://youtu.be/bxx3bM5Pm6c](https://youtu.be/bxx3bM5Pm6c)
- Github: [https://github.com/bitml-lang](https://github.com/bitml-lang)