Developing secure Bitcoin contracts with BitML

Stefano Lande

Nicola Atzei

Massimo Bartoletti

University of Cagliari

Nobuko Yoshida

Imperial College London

Roberto Zunino

University of Trento

Contracts as programs *vs* contracts as protocols

	Contracts as programs	**Contracts as protocols**
Complexity of blockchain design	**High** (gas, VM, compilers,…)	**Low** (well understood security)
Ease of programming	**High** (Solidity, …)	**Low** (Protocols + redeem scripts)
Automatic verification	**Yes** (sound ⇒ not complete)	**No** (1 contract → 1 proof)

Can we get the best of both?

(*without* creating a new coin)

Smart contracts on Bitcoin

Commit

```
out: 1 BTC:
     fun x σ . ( H(x)=h and ver_A(σ) )
     or afterAbs t: ver_B(σ)
```

Reveal

```
wit: s sig_A(Reveal)
```

Timeout

```
wit:  * sig_B(Timeout)
absLock: t
```

Pre-condition:

1) The key pair of C is $\widetilde{C}$ and the key pair of each P_i is $\widetilde{P}_i$.
2) The Ledger contains n unredeemed transactions $U_1^{\mathsf{C}}, \ldots, U_n^{\mathsf{C}}$, which can be redeemed with key $\widetilde{C}$, each having value d ₿.

The $\mathsf{CS.Commit}(\mathsf{C}, d, t, s)$ phase

3) The Committer C computes $h = H(s)$. He sends to the Ledger the transactions $Commit_1, \ldots, Commit_n$. This obviously means that he reveals h, as it is a part of each $Commit_i$.
4) If within time $\max_{\mathsf{Ledger}}$ some of the $Commit_i$ transactions does not appear on the Ledger, or if they look incorrect (e.g. they differ in the h value) then the parties abort.
5) The Committer C creates the bodies of the transactions $PayDeposit_1, \ldots, PayDeposit_n$, signs them and for all i sends the signed body $[PayDeposit_i]$ to P_i. If an appropriate transaction does not arrive to P_i, then he halts.

The $\mathsf{CS.Open}(\mathsf{C}, d, t, s)$ phase

6) The Committer C sends to the Ledger the transactions $Open_1, \ldots, Open_n$, what reveals the secret s.
7) If within time t the transaction $Open_i$ does not appear on the Ledger then P_i signs and sends the transaction $PayDeposit_i$ to the Ledger and earns d ₿.

Protocol quoted from: Andrychowicz et al. "Secure multiparty computations on Bitcoin."

Languages for Bitcoin scripts

Balzac (UniCa)

```
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)

```
or(
  and(pk(A), sha256(H)),
  and(pk(B), after(deadline))
)
```

Ivy (Chain)

```
contract Commit(kApub,kBpub: PublicKey,
                deadline: Time,
                h: Sha256(Bytes),
                v: Value) {
  clause reveal(s: Bytes, x: Signature) {
    verify sha256(s) == h
    verify checkSig(kApub, x)
    unlock v
  }
  clause timeout(x: Signature) {
    verify after(deadline)
    verify checkSig(kBpub, x)
    unlock v
  }
}
```

BitML: Bitcoin Modelling Language

$C ::= D_1 + \cdots + D_n$	contract
$D ::=$	guarded contract
withdraw A	transfer balance to A
split $v_1 \rightarrow C_1 \mid \cdots \mid v_n \rightarrow C_n$	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 … if p . C	reveal secrets a, b, …

A basic example

Precondition: A must put a 1฿:

```
{A:!1฿}
```

Contract:

```
PayOrRefund = A:withdraw B + B: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 0.2Ƀ)

```
Escrow =     A:withdraw B + B:withdraw A
           + A:Resolve    + B:Resolve

Resolve = split
              0.2Ƀ → withdraw M
          |   0.8Ƀ → M:withdraw A + M:withdraw B
```

The timed commitment in BitML

Precondition:

```
{A:!1B | A:secret a}
```

Contract:

```
  reveal a. withdraw A
+ after t : withdraw B
```

The compiled timed commitment

Tinit

out: 1 BTC:
 fun x σ σ' . (H(x)=h and $\text{ver}_{A,B}$(σ,σ'))
 or afterAbs t: $\text{ver}_{A,B}$(σ,σ')

TA

wit: s sig_A sig_B
out: 1 BTC:
 fun σ . ver_A(σ)

TB

wit: 0 sig_A sig_B
out: 1 BTC:
 fun σ . ver_B(σ)
absLock: t

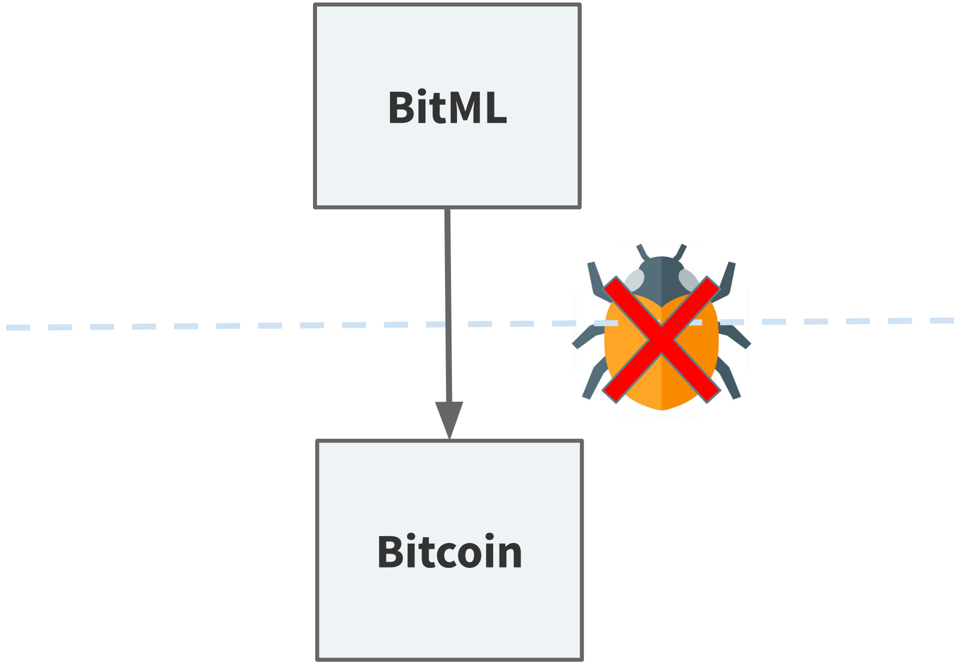
A 2-players lottery

```
{A:!3Ƀ | A:secret a | B:!3Ƀ | B:secret b}

split
 2Ƀ → reveal b . withdraw B
    + after t  : withdraw A
|2Ƀ → reveal a . withdraw A
    + after t  : withdraw B
|2Ƀ → reveal a b if a=b . withdraw A
    + reveal a b if a≠b . withdraw B
```

A 2-players lottery (fair version)

```
{A:!3Ƀ | A:secret a | B:!3Ƀ | B:secret b}

split
 2Ƀ → reveal b if 0≤b≤1 . withdraw B
    + after t  : withdraw A
|2Ƀ → reveal a . withdraw A
    + after t  : withdraw B
|2Ƀ → reveal a b if a=b . withdraw A
    + reveal a b if a≠b . withdraw B
```

Compiler security



Verification

BitML supports the automatic verification of contract properties.

- Contract-dependent properties (expressed as LTL formulae)

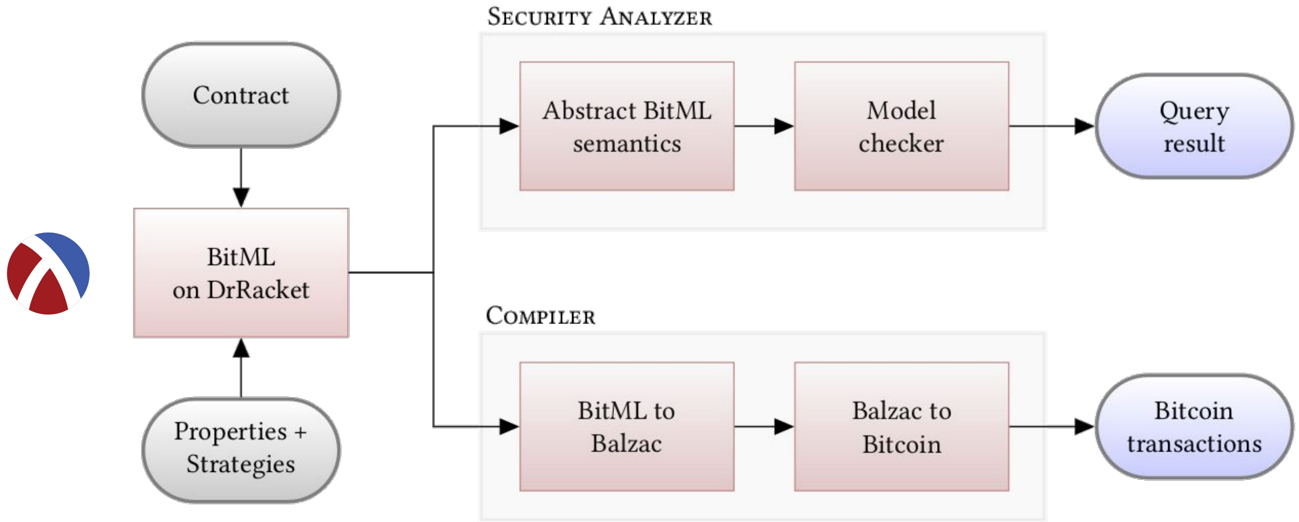
  TimedCommitment $\models$ $\square\lozenge$(**B knows a** or **B has 1 BTC**)

- **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

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

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 `add 1 BTC. C`
- Relative timelocks `in 3 days: C`
- Dynamic stipulation of subcontracts `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

Thank you

BitML toolchain

lande@unica.it

Balzac online editor

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

- Paper: https://arxiv.org/abs/1905.07639
- Tutorial: https://blockchain.unica.it/bitml
- Demo: https://youtu.be/bxx3bM5Pm6c
- Github: https://github.com/bitml-lang