Hawk: The Bolckchain Model of Cryptography and Privacy-Preserving Smart Contracts

2016 IEEE Symposium on Security and Privacy

目录

content

01. 绪论
Introduction

02. 研究方法与思路
Research methods and ideas

03. 关键技术与难点
Key Technologies and Difficulties

04. 研究成果与应用
Research Achievements and Applications

05. 论文总结
Conclusion

01

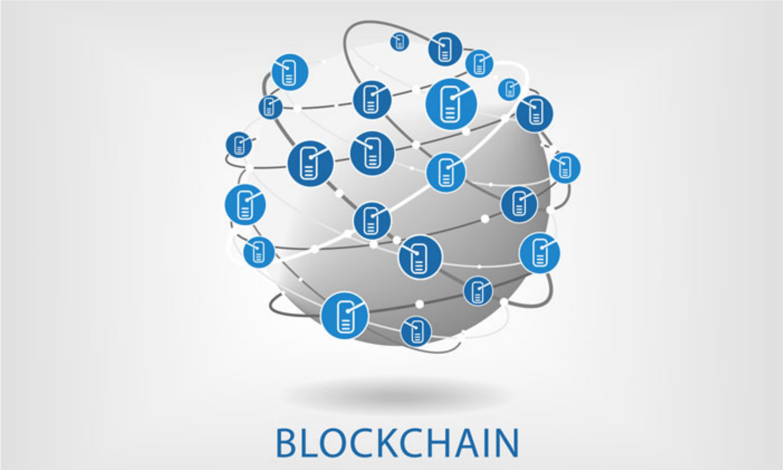
绪论

Introduction

01 绪论

Bitcoin

altcoin



ethereum

Homestead Documentation

02

研究方法与思路

Research methods and ideas

Hawk, a framework for building privacypreserving smart contracts.

There has been progress in designing privacy-preserving cryptocurrencies such as Zerocash.

Zerocash

- Private portion
- Public portion

- the blockchain’s program which will be executed by all consensus nodes;
- a program to be executed by the users; and
- a program to be executed by a special facilitating party called the manager which will be explained shortly.

Hawk

With Hawk, a non-specialist programmer can easily write a Hawk program without having to implement any cryptography.

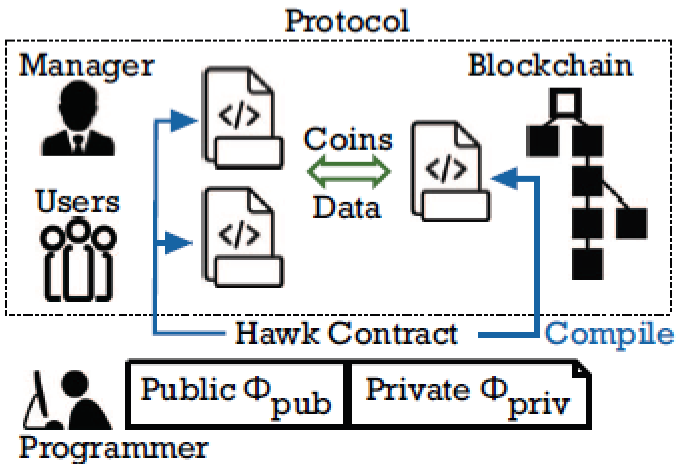


Fig. 1. Hawk overview.

Hawk overview

Security guarantees

- On-chain privacy
- Contractual security

Minimally trusted manager

even when the manager *can deviate arbitrarily from the protocol or collude with the parties, the manager cannot affect the correct execution of the contract.*

Terminology

This paper refers to the entire protocol defined by the Hawk program as a contract; and the blockchain's program is a constituent of the bigger protocol

Example: Sealed Auction

Example program

The blockchain is trusted for correctness and availability, but not trusted for privacy.

Contractual security requirement

- Input independent privacy
- Posterior privacy
- Financial fairness
- Security against a dishonest manager

Aborting and timeout

- T1 : The Hawk contract stops collecting bids after T1.
- T2 : All users should have opened their bids to the manager within T2; if a user submitted a bid but fails to open by T2, its input bid is treated as 0 (and any other potential input data treated as ⊥), such that the manager can continue.
- T3 : If the manager aborts, users can reclaim their private bids after time T3.

```
HawkDeclareParties(Seller,/* N parties */);
HawkDeclareTimeouts(/* hardcoded timeouts */);

  // Private portion ϕ_priv
private contract auction(Inp &in, Outp &out) {
  int winner = -1;
  int bestprice = -1;
  int secondprice = -1;

  for (int i = 0; i < N; i++) {
    if (in.party[i].$val > bestprice) {
      secondprice = bestprice;
      bestprice = in.party[i].$val;
      winner = i;
    } else if (in.party[i].$val > secondprice) {
      secondprice = in.party[i].$val;
    }
  }

  // Winner pays secondprice to seller
  // Everyone else is refunded
  out.Seller.$val = secondprice;
  out.party[winner].$val = bestprice-secondprice;
  out.winner = winner;
  for (int i = 0; i < N; i++) {
    if (i != winner)
      out.party[i].$val = in.party[i].$val;
  }
}

  // Public portion ϕ_pub
public contract deposit {
  // Manager deposited $N earlier
  def check():  // invoked on contract completion
    send $N to Manager  // refund manager
  def managerTimeOut():
    for (i in range($N)):
      send $1 to party[i]
}
```

Fig. 2. Hawk program for a second-price sealed auction. Code described in this paper is an approximation of our real implementation. In the public contract, the syntax "send \$N to P" corresponds to the following semantics in our cryptographic formalism: ledger[P] := ledger[P] + \$N – see Section II-B.

03

关键技术与难点

Key Technologies and Difficulties

The Blockchain model of cryptography

A The Blockchain Model

The blockchain is trusted for correctness and availability, but not trusted for privacy.

- Time *round*
- Public state
- Message delivery
- Pseudonyms
- Correctness and availability

B Formally Modeling the Blockchain

- The ideal wrapper F(·) transforms an ideal program IdealP into a UC ideal functionality F(IdealP).
- The blockchain wrapper G(·) transforms a blockchain program B to a blockchain functionality G(B). The blockchain functionality G(B) models the program executing on the blockchain.
- The protocol wrapper Π(·) transforms a user/manager program UserP into a user-side or manager-side protocol Π(UserP).

C conventions for Writing Programs

- Timer activation points
- Delayed processing in ideal programs

$\mathsf{IdealP}_{\mathsf{cash}}$

Init: Coins: a multiset of coins, each of the form $(\mathcal{P}, \$\mathsf{val})$

Mint: Upon receiving $(\mathtt{mint}, \$\mathsf{val})$ from some $\mathcal{P}$:

send $(\mathtt{mint}, \mathcal{P}, \$\mathsf{val})$ to $\mathcal{A}$

assert $\mathsf{ledger}[\mathcal{P}] \geq \$\mathsf{val}$
$\mathsf{ledger}[\mathcal{P}] := \mathsf{ledger}[\mathcal{P}] - \$\mathsf{val}$
append $(\mathcal{P}, \$\mathsf{val})$ to Coins

Pour: On $(\mathsf{pour}, \$\mathsf{val}_1, \$\mathsf{val}_2, \mathcal{P}_1, \mathcal{P}_2, \$\mathsf{val}'_1, \$\mathsf{val}'_2)$ from $\mathcal{P}$:

assert $\$\mathsf{val}_1 + \$\mathsf{val}_2 = \$\mathsf{val}'_1 + \$\mathsf{val}'_2$
if $\mathcal{P}$ is honest,
assert $(\mathcal{P}, \$\mathsf{val}_i) \in$ Coins for $i \in \{1, 2\}$
assert $\mathcal{P}_i \neq \perp$ for $i \in \{1, 2\}$
remove one $(\mathcal{P}, \$\mathsf{val}_i)$ from Coins for $i \in \{1, 2\}$
for $i \in \{1, 2\}$, if $\mathcal{P}_i$ is corrupted, send $(\mathsf{pour}, i, \mathcal{P}_i, \$\mathsf{val}'_i)$ to $\mathcal{A}$; else send $(\mathsf{pour}, i, \mathcal{P}_i)$ to $\mathcal{A}$

if $\mathcal{P}$ is corrupted:
assert $(\mathcal{P}, \$\mathsf{val}_i) \in$ Coins for $i \in \{1, 2\}$
remove one $(\mathcal{P}, \$\mathsf{val}_i)$ from Coins for $i \in \{1, 2\}$
for $i \in \{1, 2\}$: add $(\mathcal{P}_i, \$\mathsf{val}'_i)$ to Coins
for $i \in \{1, 2\}$: if $\mathcal{P}_i \neq \perp$, send $(\mathsf{pour}, \$\mathsf{val}'_i)$ to $\mathcal{P}_i$

Fig. 3. Definition of $\mathsf{IdealP}_{\mathsf{cash}}$. Notation: ledger denotes the public ledger, and Coins denotes the private pool of coins. As mentioned in Section II-C, gray background denotes batched and delayed activation. All party names correspond to pseudonyms due to notations and conventions defined in Section II-B.

Cryptography Abstractions

Overview

- Private ledger and currency transfer.
- Hawk-specific primitives.

A

Private Cash Specification $IdealP_{cash}$

- Mint.
- Pour.
- Privacy. We stress that as long as pour hides the sender, this “breaks” the transaction graph, thus preventing linking analysis.

B

Hawk Specification $IdealP_{hawk}$

- Freeze.
- Compute.
- Financial.
- Interaction with public contract.
- Security and privacy requirements.
- Timing and aborts.
- Simplifying assumptions.

$\mathsf{IdealP}_{\mathsf{cash}}$

Init: Coins: a multiset of coins, each of the form $(\mathcal{P}, \$\mathsf{val})$

Mint: Upon receiving $(\mathtt{mint}, \$\mathsf{val})$ from some $\mathcal{P}$:
send $(\mathtt{mint}, \mathcal{P}, \$\mathsf{val})$ to $\mathcal{A}$
assert $\mathsf{ledger}[\mathcal{P}] \geq \$\mathsf{val}$
$\mathsf{ledger}[\mathcal{P}] := \mathsf{ledger}[\mathcal{P}] - \$\mathsf{val}$
append $(\mathcal{P}, \$\mathsf{val})$ to Coins

Pour: On $(\mathsf{pour}, \$\mathsf{val}_1, \$\mathsf{val}_2, \mathcal{P}_1, \mathcal{P}_2, \$\mathsf{val}'_1, \$\mathsf{val}'_2)$ from $\mathcal{P}$:
assert $\$\mathsf{val}_1 + \$\mathsf{val}_2 = \$\mathsf{val}'_1 + \$\mathsf{val}'_2$
if $\mathcal{P}$ is honest,
assert $(\mathcal{P}, \$\mathsf{val}_i) \in$ Coins for $i \in \{1, 2\}$
assert $\mathcal{P}_i \neq \perp$ for $i \in \{1, 2\}$
remove one $(\mathcal{P}, \$\mathsf{val}_i)$ from Coins for $i \in \{1, 2\}$
for $i \in \{1, 2\}$, if $\mathcal{P}_i$ is corrupted, send $(\mathsf{pour}, i, \mathcal{P}_i, \$\mathsf{val}'_i)$ to $\mathcal{A}$; else send $(\mathsf{pour}, i, \mathcal{P}_i)$ to $\mathcal{A}$
if $\mathcal{P}$ is corrupted:
assert $(\mathcal{P}, \$\mathsf{val}_i) \in$ Coins for $i \in \{1, 2\}$
remove one $(\mathcal{P}, \$\mathsf{val}_i)$ from Coins for $i \in \{1, 2\}$
for $i \in \{1, 2\}$: add $(\mathcal{P}_i, \$\mathsf{val}'_i)$ to Coins
for $i \in \{1, 2\}$: if $\mathcal{P}_i \neq \perp$, send $(\mathsf{pour}, \$\mathsf{val}'_i)$ to $\mathcal{P}_i$

Fig. 3. Definition of $\mathsf{IdealP}_{\mathsf{cash}}$. Notation: ledger denotes the public ledger, and Coins denotes the private pool of coins. As mentioned in Section II-C, gray background denotes batched and delayed activation. All party names correspond to pseudonyms due to notations and conventions defined in Section II-B.

Cryptography Abstractions

Overview

- Private ledger and currency transfer.
- Hawk-specific primitives.

A

Private Cash Specification $IdealP_{cash}$

- Mint.
- Pour.
- Privacy. We stress that as long as pour hides the sender, this “breaks” the transaction graph, thus preventing linking analysis.

B

Hawk Specification $IdealP_{hawk}$

- Freeze.
- Compute.
- Financial.
- Interaction with public contract.
- Security and privacy requirements.
- Timing and aborts.
- Simplifying assumptions.

$\mathsf{IdealP}_{\mathbf{hawk}}(\mathcal{P}_{\mathcal{M}}, \{\mathcal{P}_i\}_{i\in[N]}, T_1, T_2, \phi_{\mathbf{priv}}, \phi_{\mathbf{pub}})$

Init: Call $\mathsf{IdealP}_{\mathsf{cash}}$.**Init**. Additionally:

FrozenCoins: a set of coins and private inputs received by this contract, each of the form $(\mathcal{P}, \mathsf{in}, \$\mathsf{val})$. Initialize FrozenCoins := $\emptyset$.

Freeze: Upon receiving $(\mathtt{freeze}, \$\mathsf{val}_i, \mathsf{in}_i)$ from $\mathcal{P}_i$ for some $i \in [N]$:

assert current time $T < T_1$
assert $\mathcal{P}_i$ has not called freeze earlier.
assert at least one copy of $(\mathcal{P}_i, \$\mathsf{val}_i) \in$ Coins
send $(\mathtt{freeze}, \mathcal{P}_i)$ to $\mathcal{A}$

add $(\mathcal{P}_i, \$\mathsf{val}_i, \mathsf{in}_i)$ to FrozenCoins
remove one $(\mathcal{P}_i, \$\mathsf{val}_i)$ from Coins

Compute: Upon receiving compute from $\mathcal{P}_i$ for some $i \in [N]$:

assert current time $T_1 \le T < T_2$
if $\mathcal{P}_{\mathcal{M}}$ is corrupted, send $(\mathtt{compute}, \mathcal{P}_i, \$\mathsf{val}_i, \mathsf{in}_i)$ to $\mathcal{A}$
else send $(\mathtt{compute}, \mathcal{P}_i)$ to $\mathcal{A}$

let $(\mathcal{P}_i, \$\mathsf{val}_i, \mathsf{in}_i)$ be the item in FrozenCoins corresponding to $\mathcal{P}_i$
send $(\mathtt{compute}, \mathcal{P}_i, \$\mathsf{val}_i, \mathsf{in}_i)$ to $\mathcal{P}_{\mathcal{M}}$

Finalize: Upon receiving $(\mathtt{finalize}, \mathsf{in}_{\mathcal{M}}, \mathsf{out})$ from $\mathcal{P}_{\mathcal{M}}$:

assert current time $T \ge T_2$
assert $\mathcal{P}_{\mathcal{M}}$ has not called finalize earlier
for $i \in [N]$:
  let $(\$\mathsf{val}_i, \mathsf{in}_i) := (0, \perp)$ if $\mathcal{P}_i$ has not called compute
$(\{\$\mathsf{val}'_i\}, \mathsf{out}^{\dagger}) := \phi_{\mathsf{priv}}(\{\$\mathsf{val}_i, \mathsf{in}_i\}, \mathsf{in}_{\mathcal{M}})$
assert $\mathsf{out}^{\dagger} = \mathsf{out}$
assert $\sum_{i\in[N]} \$\mathsf{val}_i = \sum_{i\in[N]} \$\mathsf{val}'_i$
send $(\mathtt{finalize}, \mathsf{in}_{\mathcal{M}}, \mathsf{out})$ to $\mathcal{A}$
for each corrupted $\mathcal{P}_i$ that called compute: send $(\mathcal{P}_i, \$\mathsf{val}'_i)$ to $\mathcal{A}$

call $\phi_{\mathsf{pub}}.\mathsf{check}(\mathsf{in}_{\mathcal{M}}, \mathsf{out})$
for $i \in [N]$ such that $\mathcal{P}_i$ called compute:
  add $(\mathcal{P}_i, \$\mathsf{val}'_i)$ to Coins
  send $(\mathtt{finalize}, \$\mathsf{val}'_i)$ to $\mathcal{P}_i$

ϕ_{pub}: Run a local instance of public contract ϕ_{pub}. Messages between the adversary to ϕ_{pub}, and from ϕ_{pub} to parties are forwarded directly.
Upon receiving message (pub, m) from party $\mathcal{P}$:

notify $\mathcal{A}$ of (pub, m)

send m to ϕ_{pub} on behalf of $\mathcal{P}$

$\mathsf{IdealP}_{\mathsf{cash}}$: include $\mathsf{IdealP}_{\mathsf{cash}}$ (Figure 3).

Fig. 4. Definition of $\mathsf{IdealP}_{\mathsf{hawk}}$. Notations: FrozenCoins denotes frozen coins owned by the contract; Coins denotes the global private coin pool defined by $\mathsf{IdealP}_{\mathsf{cash}}$; and $(\mathsf{in}_i, \mathsf{val}_i)$ denotes the input data and frozen coin value of party $\mathcal{P}_i$.

Cryptographic Protocols

A warmup: private cash and money transfers

- Existence of coins being spent
- No double spending.
- Money conservation.

B Binding Privacy and Programmable Logic

- Freeze
- Compute.
- Finalize.

C Extension and Discussions

- Refunding frozen coins to users.
- Instantiating the manager with trusted hardware.
- Pouring anonymously to long-lived pseudonyms.
- Open enrollment of pseudonyms.

Protocol $\mathsf{UserP_{cash}}$

Init: Wallet: stores $\mathcal{P}$'s spendable coins, initially $\emptyset$

GenNym: sample a random seed $\mathsf{sk_{prf}}$
$\mathsf{pk_{prf}} := \mathsf{PRF}_{\mathsf{sk_{prf}}}(0)$
return $\mathsf{pk_{prf}}$

Mint: On input $(\mathtt{mint}, \$\mathsf{val})$,
sample a commitment randomness s
$\mathsf{coin} := \mathsf{Comm}_s(\$\mathsf{val})$
store $(s, \$\mathsf{val}, \mathsf{coin})$ in Wallet
send $(\mathtt{mint}, \$\mathsf{val}, s)$ to $\mathcal{G}(\mathsf{Blockchain_{cash}})$

Pour (as sender): On input $(\mathtt{pour}, \$\mathsf{val}_1, \$\mathsf{val}_2, \mathcal{P}_1, \mathcal{P}_2, \$\mathsf{val}'_1, \$\mathsf{val}'_2)$,
assert $\$\mathsf{val}_1 + \$\mathsf{val}_2 = \$\mathsf{val}'_1 + \$\mathsf{val}'_2$
for $i \in \{1,2\}$, assert $(s_i, \$\mathsf{val}_i, \mathsf{coin}_i) \in$ Wallet for some (s_i, coin_i)
let MT be a merkle tree over $\mathsf{Blockchain_{cash}}$.Coins
for $i \in \{1,2\}$:
remove one $(s_i, \$\mathsf{val}_i, \mathsf{coin}_i)$ from Wallet
$\mathsf{sn}_i := \mathsf{PRF}_{\mathsf{sk_{prf}}}(\mathcal{P} \| \mathsf{coin}_i)$
let branch_i be the branch of $(\mathcal{P}, \mathsf{coin}_i)$ in MT
sample randomness s'_i, r_i
$\mathsf{coin}'_i := \mathsf{Comm}_{s'_i}(\$\mathsf{val}'_i)$
$\mathsf{ct}_i := \mathsf{ENC}(\mathcal{P}_i.\mathsf{epk}, r_i, \$\mathsf{val}'_i \| s'_i)$
$\mathsf{statement} := (\mathsf{MT.root}, \{\mathsf{sn}_i, \mathcal{P}_i, \mathsf{coin}'_i\}_{i\in\{1,2\}})$
$\mathsf{witness} := (\mathcal{P}, \mathsf{sk_{prf}}, \{\mathsf{branch}_i, s_i, \$\mathsf{val}_i, s'_i, r_i, \$\mathsf{val}'_i\})$
$\pi := \mathsf{NIZK.Prove}(\mathcal{L}_{\mathtt{POUR}}, \mathsf{statement}, \mathsf{witness})$
$\mathsf{AnonSend}(\mathtt{pour}, \pi, \{\mathsf{sn}_i, \mathcal{P}_i, \mathsf{coin}'_i, \mathsf{ct}_i\}_{i\in\{1,2\}})$ to $\mathcal{G}(\mathsf{Blockchain_{cash}})$

Pour (as recipient): On receive $(\mathtt{pour}, \mathsf{coin}, \mathsf{ct})$ from $\mathcal{G}(\mathsf{Blockchain_{cash}})$:
let $(\$\mathsf{val} \| s) := \mathsf{DEC}(\mathsf{esk}, \mathsf{ct})$
assert $\mathsf{Comm}_s(\$\mathsf{val}) = \mathsf{coin}$
store $(s, \$\mathsf{val}, \mathsf{coin})$ in Wallet
output $(\mathtt{pour}, \$\mathsf{val})$

$\mathsf{Blockchain_{cash}}$

Init: crs: a reference string for the underlying NIZK system
Coins: a set of coin commitments, initially $\emptyset$
SpentCoins: set of spent serial numbers, initially $\emptyset$

Mint: Upon receiving $(\mathtt{mint}, \$\mathsf{val}, s)$ from some party $\mathcal{P}$,
$\mathsf{coin} := \mathsf{Comm}_s(\$\mathsf{val})$
assert $(\mathcal{P}, \mathsf{coin}) \notin$ Coins
assert $\mathsf{ledger}[\mathcal{P}] \geq \$\mathsf{val}$
$\mathsf{ledger}[\mathcal{P}] := \mathsf{ledger}[\mathcal{P}] - \$\mathsf{val}$
add $(\mathcal{P}, \mathsf{coin})$ to Coins

Pour: Anonymous receive $(\mathtt{pour}, \pi, \{\mathsf{sn}_i, \mathcal{P}_i, \mathsf{coin}_i, \mathsf{ct}_i\}_{i\in\{1,2\}})$
let MT be a merkle tree built over Coins
$\mathsf{statement} := (\mathsf{MT.root}, \{\mathsf{sn}_i, \mathcal{P}_i, \mathsf{coin}_i\}_{i\in\{1,2\}})$
assert $\mathsf{NIZK.Verify}(\mathcal{L}_{\mathtt{POUR}}, \pi, \mathsf{statement})$
for $i \in \{1,2\}$,
assert $\mathsf{sn}_i \notin$ SpentCoins
assert $(\mathcal{P}_i, \mathsf{coin}_i) \notin$ Coins
add sn_i to SpentCoins
add $(\mathcal{P}_i, \mathsf{coin}_i)$ to Coins
send $(\mathtt{pour}, \mathsf{coin}_i, \mathsf{ct}_i)$ to $\mathcal{P}_i$,

Relation $(\mathsf{statement}, \mathsf{witness}) \in \mathcal{L}_{\mathtt{POUR}}$ is defined as:
parse statement as $(\mathsf{MT.root}, \{\mathsf{sn}_i, \mathcal{P}_i, \mathsf{coin}'_i\}_{i\in\{1,2\}})$
parse witness as $(\mathcal{P}, \mathsf{sk_{prf}}, \{\mathsf{branch}_i, s_i, \$\mathsf{val}_i, s'_i, r_i, \$\mathsf{val}'_i\})$
assert $\mathcal{P}.\mathsf{pk_{prf}} = \mathsf{PRF}_{\mathsf{sk_{prf}}}(0)$
assert $\$\mathsf{val}_1 + \$\mathsf{val}_2 = \$\mathsf{val}'_1 + \$\mathsf{val}'_2$
for $i \in \{1,2\}$,
$\mathsf{coin}_i := \mathsf{Comm}_{s_i}(\$\mathsf{val}_i)$
assert $\mathsf{MerkleBranch}(\mathsf{MT.root}, \mathsf{branch}_i, (\mathcal{P} \| \mathsf{coin}_i))$
assert $\mathsf{sn}_i = \mathsf{PRF}_{\mathsf{sk_{prf}}}(\mathcal{P} \| \mathsf{coin}_i)$
assert $\mathsf{coin}'_i = \mathsf{Comm}_{s'_i}(\$\mathsf{val}'_i)$

Fig. 5. $\mathsf{UserP_{cash}}$ construction. A trusted setup phase generates the NIZK's common reference string crs. For notational convenience, we omit writing the crs explicitly in the construction. The Merkle tree MT is stored on the blockchain and not computed on the fly – we omit stating this in the protocol for notational simplicity. The protocol wrapper $\Pi(\cdot)$ invokes **GenNym** whenever a party creates a new pseudonym.

$\mathsf{Blockchain}_{\mathsf{hawk}}(\mathcal{P}_{\mathcal{M}}, \{\mathcal{P}_i\}_{i\in[N]}, T_1, T_2, \phi_{\mathsf{priv}}, \phi_{\mathsf{pub}})$

Init: See $\mathsf{IdealP}_{\mathsf{hawk}}$ for description of parameters
Call $\mathsf{Blockchain}_{\mathsf{cash}}$.**Init**.

Freeze: Upon receiving $(\mathtt{freeze}, \pi, \mathsf{sn}_i, \mathsf{cm}_i)$ from $\mathcal{P}_i$:
- assert current time $T \le T_1$
- assert this is the first $\mathtt{freeze}$ from $\mathcal{P}_i$
- let MT be a merkle tree built over Coins
- assert $\mathsf{sn}_i \notin \mathsf{SpentCoins}$
- $\mathsf{statement} := (\mathcal{P}_i, \mathsf{MT.root}, \mathsf{sn}_i, \mathsf{cm}_i)$
- assert $\mathsf{NIZK.Verify}(\mathcal{L}_{\mathtt{FREEZE}}, \pi, \mathsf{statement})$
- add sn_i to SpentCoins and store cm_i for later

Compute: Upon receiving $(\mathtt{compute}, \pi, \mathsf{ct})$ from $\mathcal{P}_i$:
- assert $T_1 \le T < T_2$ for current time T
- assert $\mathsf{NIZK.Verify}(\mathcal{L}_{\mathtt{COMPUTE}}, \pi, (\mathcal{P}_{\mathcal{M}}, \mathsf{cm}_i, \mathsf{ct}))$
- send $(\mathtt{compute}, \mathcal{P}_i, \mathsf{ct})$ to $\mathcal{P}_{\mathcal{M}}$

Finalize: On receiving $(\mathtt{finalize}, \pi, \mathsf{in}_{\mathcal{M}}, \mathsf{out}, \{\mathsf{coin}'_i, \mathsf{ct}_i\}_{i\in[N]})$ from $\mathcal{P}_{\mathcal{M}}$:
- assert current time $T \ge T_2$
- for every $\mathcal{P}_i$ that has not called $\mathtt{compute}$, set $\mathsf{cm}_i := \perp$
- $\mathsf{statement} := (\mathsf{in}_{\mathcal{M}}, \mathsf{out}, \{\mathsf{cm}_i, \mathsf{coin}'_i, \mathsf{ct}_i\}_{i\in[N]})$
- assert $\mathsf{NIZK.Verify}(\mathcal{L}_{\mathtt{FINALIZE}}, \pi, \mathsf{statement})$
- for $i \in [N]$:
  - assert $\mathsf{coin}'_i \notin \mathsf{Coins}$
  - add coin'_i to Coins
  - send $(\mathtt{finalize}, \mathsf{coin}'_i, \mathsf{ct}_i)$ to $\mathcal{P}_i$
- Call $\phi_{\mathsf{pub}}.\mathsf{check}(\mathsf{in}_{\mathcal{M}}, \mathsf{out})$

$\mathsf{Blockchain}_{\mathsf{cash}}$: include $\mathsf{Blockchain}_{\mathsf{cash}}$
ϕ_{pub} : include user-defined public contract ϕ_{pub}

Relation $(\mathsf{statement}, \mathsf{witness}) \in \mathcal{L}_{\mathtt{FREEZE}}$ is defined as:
- parse statement as $(\mathcal{P}, \mathsf{MT.root}, \mathsf{sn}, \mathsf{cm})$
- parse witness as $(\mathsf{coin}, \mathsf{sk}_{\mathsf{prf}}, \mathsf{branch}, s, \$\mathsf{val}, \mathsf{in}, k, s')$
- $\mathsf{coin} := \mathsf{Comm}_s(\$\mathsf{val})$
- assert $\mathsf{MerkleBranch}(\mathsf{MT.root}, \mathsf{branch}, (\mathcal{P}\|\mathsf{coin}))$
- assert $\mathcal{P}.\mathsf{pk}_{\mathsf{prf}} = \mathsf{sk}_{\mathsf{prf}}(0)$
- assert $\mathsf{sn} = \mathsf{PRF}_{\mathsf{sk}_{\mathsf{prf}}}(\mathcal{P}\|\mathsf{coin})$
- assert $\mathsf{cm} = \mathsf{Comm}_{s'}(\$\mathsf{val}\|\mathsf{in}\|k)$

Relation $(\mathsf{statement}, \mathsf{witness}) \in \mathcal{L}_{\mathtt{COMPUTE}}$ is defined as:
- parse statement as $(\mathcal{P}_{\mathcal{M}}, \mathsf{cm}, \mathsf{ct})$
- parse witness as $(\$\mathsf{val}, \mathsf{in}, k, s', r)$
- assert $\mathsf{cm} = \mathsf{Comm}_{s'}(\$\mathsf{val}\|\mathsf{in}\|k)$
- assert $\mathsf{ct} = \mathsf{ENC}(\mathcal{P}_{\mathcal{M}}.\mathsf{epk}, r, (\$\mathsf{val}\|\mathsf{in}\|k\|s'))$

Relation $(\mathsf{statement}, \mathsf{witness}) \in \mathcal{L}_{\mathtt{FINALIZE}}$ is defined as:
- parse statement as $(\mathsf{in}_{\mathcal{M}}, \mathsf{out}, \{\mathsf{cm}_i, \mathsf{coin}'_i, \mathsf{ct}_i\}_{i\in[N]})$
- parse witness as $\{s_i, \$\mathsf{val}_i, \mathsf{in}_i, s'_i, k_i\}_{i\in[N]}$
- $(\{\$\mathsf{val}'_i\}_{i\in[N]}, \mathsf{out}) := \phi_{\mathsf{priv}}(\{\$\mathsf{val}_i, \mathsf{in}_i\}_{i\in[N]}, \mathsf{in}_{\mathcal{M}})$
- assert $\sum_{i\in[N]} \$\mathsf{val}_i = \sum_{i\in[N]} \$\mathsf{val}'_i$
- for $i \in [N]$:
  - assert $\mathsf{cm}_i = \mathsf{Comm}_{s_i}(\$\mathsf{val}_i\|\mathsf{in}_i\|k_i)$
    $\vee (\$\mathsf{val}_i, \mathsf{in}_i, k_i, s_i, \mathsf{cm}_i) = (0, \perp, \perp, \perp, \perp)$
  - assert $\mathsf{ct}_i = \mathsf{SENC}_{k_i}(s'_i\|\$\mathsf{val}'_i)$
  - assert $\mathsf{coin}'_i = \mathsf{Comm}_{s'_i}(\$\mathsf{val}'_i)$

Protocol $\mathsf{UserP}_{\mathsf{hawk}}(\mathcal{P}_{\mathcal{M}}, \{\mathcal{P}_i\}_{i\in[N]}, T_1, T_2, \phi_{\mathsf{priv}}, \phi_{\mathsf{pub}})$

Init: Call $\mathsf{UserP}_{\mathsf{cash}}$.**Init**.

Protocol for a party $\mathcal{P} \in \{\mathcal{P}_i\}_{i\in[N]}$:

Freeze: On input $(\mathtt{freeze}, \$\mathsf{val}, \mathsf{in})$ as party $\mathcal{P}$:
- assert current time $T < T_1$
- assert this is the first $\mathtt{freeze}$ input
- let MT be a merkle tree over $\mathsf{Blockchain}_{\mathsf{cash}}$.Coins
- assert that some entry $(s, \$\mathsf{val}, \mathsf{coin}) \in \mathsf{Wallet}$ for some (s, coin)
- remove one $(s, \$\mathsf{val}, \mathsf{coin})$ from Wallet
- $\mathsf{sn} := \mathsf{PRF}_{\mathsf{sk}_{\mathsf{prf}}}(\mathcal{P}\|\mathsf{coin})$
- let branch be the branch of $(\mathcal{P}, \mathsf{coin})$ in MT
- sample a symmetric encryption key k
- sample a commitment randomness s'
- $\mathsf{cm} := \mathsf{Comm}_{s'}(\$\mathsf{val}\|\mathsf{in}\|k)$
- $\mathsf{statement} := (\mathcal{P}, \mathsf{MT.root}, \mathsf{sn}, \mathsf{cm})$
- $\mathsf{witness} := (\mathsf{coin}, \mathsf{sk}_{\mathsf{prf}}, \mathsf{branch}, s, \$\mathsf{val}, \mathsf{in}, k, s')$
- $\pi := \mathsf{NIZK.Prove}(\mathcal{L}_{\mathtt{FREEZE}}, \mathsf{statement}, \mathsf{witness})$
- send $(\mathtt{freeze}, \pi, \mathsf{sn}, \mathsf{cm})$ to $\mathcal{G}(\mathsf{Blockchain}_{\mathsf{hawk}})$
- store in, cm, \$val, s', and k to use later (in $\mathtt{compute}$)

Compute: On input $(\mathtt{compute})$ as party $\mathcal{P}$:
- assert current time $T_1 \le T < T_2$
- sample encryption randomness r
- $\mathsf{ct} := \mathsf{ENC}(\mathcal{P}_{\mathcal{M}}.\mathsf{epk}, r, (\$\mathsf{val}\|\mathsf{in}\|k\|s'))$
- $\pi := \mathsf{NIZK.Prove}((\mathcal{P}_{\mathcal{M}}, \mathsf{cm}, \mathsf{ct}), (\$\mathsf{val}, \mathsf{in}, k, s', r))$
- send $(\mathtt{compute}, \pi, \mathsf{ct})$ to $\mathcal{G}(\mathsf{Blockchain}_{\mathsf{hawk}})$

Finalize: Receive $(\mathtt{finalize}, \mathsf{coin}, \mathsf{ct})$ from $\mathcal{G}(\mathsf{Blockchain}_{\mathsf{hawk}})$:
- decrypt $(s\|\$\mathsf{val}) := \mathsf{SDEC}_k(\mathsf{ct})$
- store $(s, \$\mathsf{val}, \mathsf{coin})$ in Wallet
- output $(\mathtt{finalize}, \$\mathsf{val})$

Protocol for manager $\mathcal{P}_{\mathcal{M}}$:

Compute: On receive $(\mathtt{compute}, \mathcal{P}_i, \mathsf{ct})$ from $\mathcal{G}(\mathsf{Blockchain}_{\mathsf{hawk}})$:
- decrypt and store $(\$\mathsf{val}_i\|\mathsf{in}_i\|k_i\|s_i) := \mathsf{DEC}(\mathsf{esk}, \mathsf{ct})$
- store $\mathsf{cm}_i := \mathsf{Comm}_{s_i}(\$\mathsf{val}_i\|\mathsf{in}_i\|k_i)$
- output $(\mathcal{P}_i, \$\mathsf{val}_i, \mathsf{in}_i)$
- If this is the last $\mathtt{compute}$ received:
  - for $i \in [N]$ such that $\mathcal{P}_i$ has not called $\mathtt{compute}$,
    $(\$\mathsf{val}_i, \mathsf{in}_i, k_i, s_i, \mathsf{cm}_i) := (0, \perp, \perp, \perp, \perp)$
  - $(\{\$\mathsf{val}'_i\}_{i\in[N]}, \mathsf{out}) := \phi_{\mathsf{priv}}(\{\$\mathsf{val}_i, \mathsf{in}_i\}_{i\in[N]}, \mathsf{in}_{\mathcal{M}})$
  - store and output $(\{\$\mathsf{val}'_i\}_{i\in[N]}, \mathsf{out})$

Finalize: On input $(\mathtt{finalize}, \mathsf{in}_{\mathcal{M}}, \mathsf{out})$:
- assert current time $T \ge T_2$
- for $i \in [N]$:
  - sample a commitment randomness s'_i
  - $\mathsf{coin}'_i := \mathsf{Comm}_{s'_i}(\$\mathsf{val}'_i)$
  - $\mathsf{ct}_i := \mathsf{SENC}_{k_i}(s'_i\|\$\mathsf{val}'_i)$
- $\mathsf{statement} := (\mathsf{in}_{\mathcal{M}}, \mathsf{out}, \{\mathsf{cm}_i, \mathsf{coin}'_i, \mathsf{ct}_i\}_{i\in[N]})$
- $\mathsf{witness} := \{s_i, \$\mathsf{val}_i, \mathsf{in}_i, s'_i, k_i\}_{i\in[N]}$
- $\pi := \mathsf{NIZK.Prove}(\mathsf{statement}, \mathsf{witness})$
- send $(\mathtt{finalize}, \pi, \mathsf{in}_{\mathcal{M}}, \mathsf{out}, \{\mathsf{coin}'_i, \mathsf{ct}_i\})$ to $\mathcal{G}(\mathsf{Blockchain}_{\mathsf{hawk}})$

$\mathsf{UserP}_{\mathsf{cash}}$: include $\mathsf{UserP}_{\mathsf{cash}}$.

Fig. 6. $\mathsf{Blockchain}_{\mathsf{hawk}}$ and $\mathsf{UserP}_{\mathsf{hawk}}$ construction.

04

研究成果与应用

Research Achievements and Applications

Adopting SNARKs in UC Protocols and Practical Optimizations

A Using SNARKs in UC Protocols

- SNARKs : Succinct Non-interactive Arguments of Knowledge
- NIZK ; non-interactive zero-knowledge
- Our implementations thus adopt the efficient SNARK-lifting transformations proposed by Kosba et al.

B Practical Considerations

- Efficient SNARK circuits
- Optimizations of finalize

  Minimize SSE-secure NIZKs

  Minimize public-key encryption in SNARKs
- Remarks about the common reference string

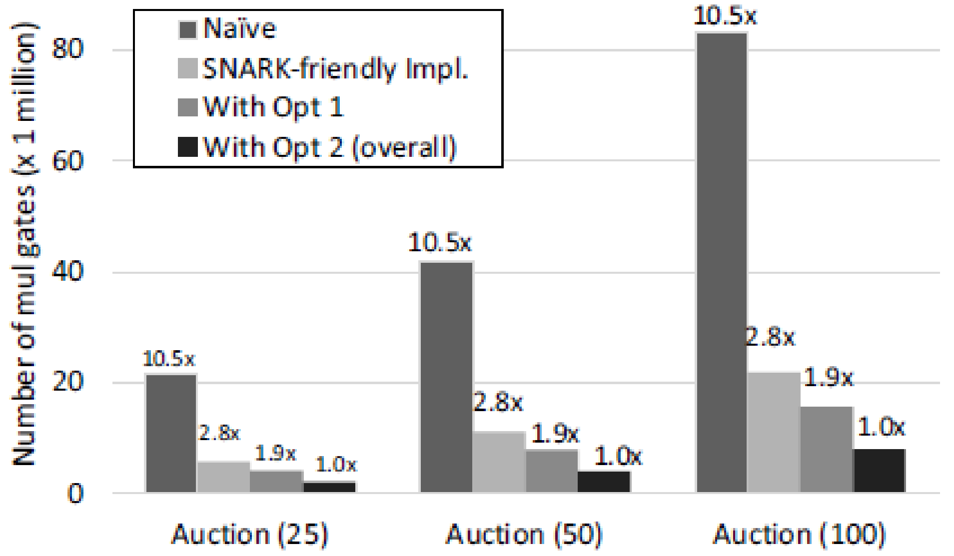


Fig. 9. **Gains after adding each optimization to the `finalize` auction circuit, with 25, 50 and 100 Bidders.** Opt 1 and Opt 2 are two practical optimizations detailed in Section V.

Compiler Implementation

A

- Preprocessing
  - Public contract
  - Private contract
- Circuit Augmentation
  - Φpriv
- Cryptographic protocol
  - Libsnark

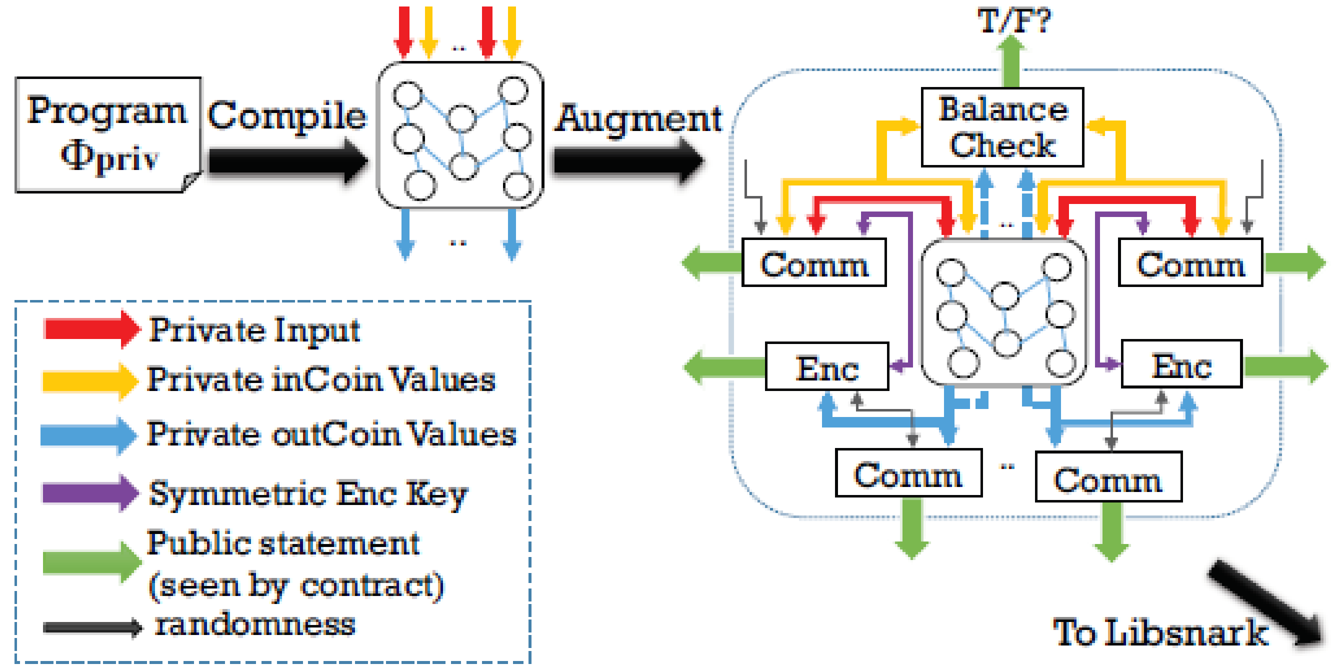


Fig. 7. Compiler overview. Circuit augmentation for `finalize`.

Performance Evaluation

B

- Amazon EC2 `r3.8xlarge` virtual machine
- A maximum of 2^{64} leaves for the Merkle trees
- 80-bit and 112-bit security levels.
- Benchmarks actually consume at most 27GB of memory and 4 cores in the most expensive case.

TABLE I
Performance of the zk-SNARK circuits for the user-side circuits: pour, freeze AND compute (SAME FOR ALL APPLICATIONS). MUL denotes multiple (4) cores, and ONE denotes a single core. The mint operation does not involve any SNARKs, and can be computed within tens of microseconds. The Proof includes any additional cryptographic material used for the SNARK-lifting transformation.

		80-bit security			112-bit security		
		pour	freeze	compute	pour	freeze	compute
KeyGen(s)	MUL	26.3	18.2	15.9	36.7	30.5	34.6
	ONE	88.2	63.3	54.42	137.2	111.1	131.8
Prove(s)	MUL	12.4	8.4	9.3	18.5	15.7	16.8
	ONE	27.5	20.7	22.5	42.2	40.5	41.7
Verify(ms)		9.7	9.1	10.0	9.9	9.3	9.9
EvalKey(MB)		148	106	90	236	189	224
VerKey(KB)		7.3	4.4	7.8	8.7	5.3	8.4
Proof(KB)		0.68	0.68	0.68	0.71	0.71	0.71
Stmt(KB)		0.48	0.16	0.53	0.57	0.19	0.53

TABLE II
Performance of the zk-SNARK circuits for the manager circuit finalize for different applications. The manager circuits are the same for both security levels. MUL denotes multiple (4) cores, and ONE denotes a single core.

		swap	rps	auction		crowdfund	
#Parties		2	2	10	100	10	100
KeyGen(s)	MUL	8.6	8.0	32.3	300.4	32.16	298.1
	ONE	27.8	24.9	124	996.3	124.4	976.5
Prove(s)	MUL	3.2	3.1	15.4	169.3	15.2	169.2
	ONE	7.6	7.4	40.1	384.2	40.3	377.5
Verify(ms)		8.4	8.4	10	19.9	10	19.8
EvalKey(GB)		0.04	0.04	0.21	1.92	0.21	1.91
VerKey(KB)		3.3	2.9	12.9	113.8	12.9	113.8
Proof(KB)		0.28	0.28	0.28	0.28	0.28	0.28
Stmt(KB)		0.22	0.2	1.03	9.47	1.03	9.47

- We highlight some important observations:
  - On-chain computation
  - On-chain public parameters
  - Manager computation
  - User computation
- Savings from protocols optimizations

Performance Evaluation

B

- We highlight some important observations:
  - On-chain computation
  - On-chain public parameters
  - Manager computation
  - User computation
- Savings from protocols optimizations

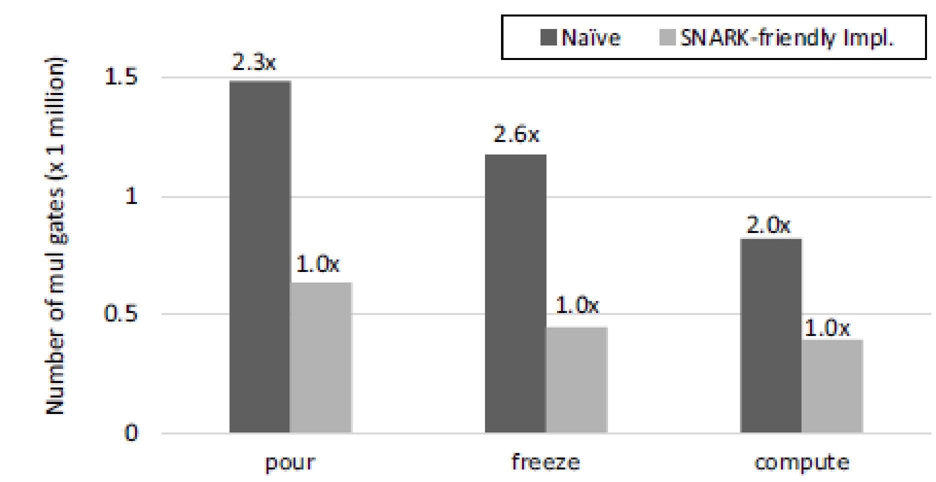


Fig. 8. **Gains of using SNARK-friendly implementation for the user-side circuits:** `pour`, `freeze` and `compute` at 80-bit security.

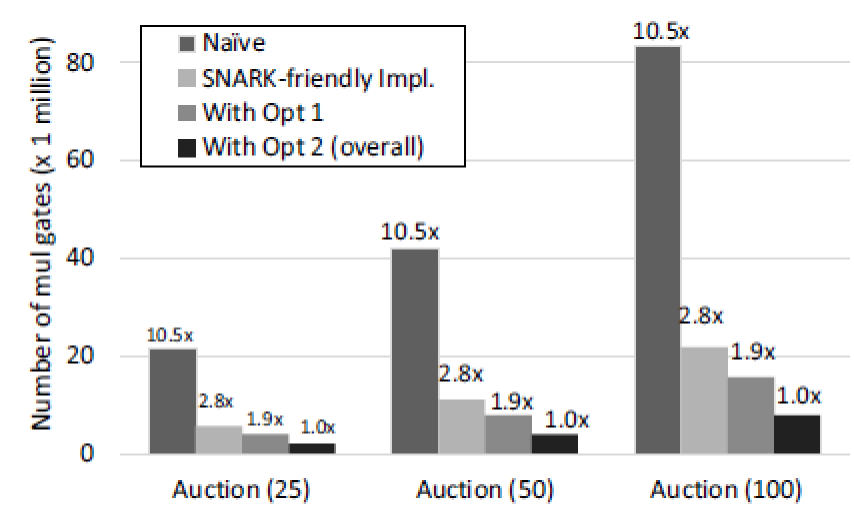


Fig. 9. **Gains after adding each optimization to the `finalize` auction circuit, with 25, 50 and 100 Bidders.** Opt 1 and Opt 2 are two practical optimizations detailed in Section V.

05

论文总结

Conclusion

Conclusion

We present Hawk, a decentralized smart contract system that does not store financial transactions in the clear on the blockchain, thus retaining transactional privacy from the public's view. A Hawk programmer can write a private smart contract in an intuitive manner without having to implement cryptography, and our compiler automatically generates an efficient cryptographic protocol where contractual parties interact with the blockchain, using cryptographic primitives such as zero-knowledge proofs.

To formally define and reason about the security of our protocols, we are the first to formalize the blockchain model of cryptography. The formal modeling is of independent interest. We advocate the community to adopt such a formal model when designing applications atop decentralized blockchains.