zkay: Specifying and Enforcing Data Privacy in Smart Contracts

Samuel Steffen
Benjamin Bichsel
Mario Gersbach
Noa Melchior
Petar Tsankov
Martin Vechev
Smart Contracts

miners execute smart contracts

smart contract

data
Smart Contracts

miners execute smart contracts

data is public

Alice’s data 42
Bob’s data 13
Privacy?

Medical data and the rise of blockchain

6th July 2018

http://www.pharmatimes.com/web_exclusives/medical_data_and_the_rise_of_blockchain_1243441
Privacy?

How Blockchain Could Give Us a Smarter Energy Grid

Energy experts believe that blockchain technology can solve a maze of red tape and data management problems.

by Mike Orcutt


---

Medical data and the rise of blockchain

6th July 2018

http://www.pharmatimes.com/web_exclusives/medical_data_and_the_rise_of_blockchain_1243441
Privacy?

Blockchain / Smart Contracts

How Blockchain Could Give Us a Smarter Energy Grid

Energy experts believe that blockchain technology can solve a maze of red tape and data management problems.

by Mike Orcutt

Oct 12, 2017


Medical data and the rise of blockchain

6th July 2018

https://www.pharmatimes.com/web_exclusives/medical_data_and_the_rise_of_blockchain_1243441

Paradigm Shift: Biometrics And The Blockchain Will Replace Paper Passports Sooner Than You Think

Suzanne Rovane Kelleher Contributor (Travel)

Biometrics and blockchain are the keys to the future of traveler identification.

Data Privacy for Smart Contracts

- Alice's data
- Bob's data
- data is private

smart contract → data

Alice's data
Bob's data
Existing Work

Zerocoin [S&P 13]
Zerocash [S&P 14]
Bolt [CCS 17]
Existing Work

Zerocoin [S&P 13]
Zerocash [S&P 14]
Bolt [CCS 17]

only for payments
Existing Work

- Zerocoin [S&P 13]
- Zerocash [S&P 14]
- Bolt [CCS 17]

- Hawk [S&P 16]
- Arbitrum [Usenix 18]
- Ekiden [EuroS&P 19]

only for payments
Existing Work

- Zerocoin [S&P 13]
- Zerocash [S&P 14]
- Bolt [CCS 17]
- Hawk [S&P 16]
- Arbitrum [Usenix 18]
- Ekiden [EuroS&P 19]

- only for payments
- introduce trusted third parties / HW
Existing Work

- Zerocoin [S&P 13]
- Zerocash [S&P 14]
- Bolt [CCS 17]
- Hawk [S&P 16]
- Arbitrum [Usenix 18]
- Ekiden [EuroS&P 19]

Our work: Leverage *cryptographic primitives* on *existing public* blockchains
Existing Work

Zerocoin [S&P 13]
Zerocash [S&P 14]
Bolt [CCS 17]

only for payments

Hawk [S&P 16]
Arbitrum [Usenix 18]
Ekiden [EuroS&P 19]

introduce trusted third parties / HW

Our work:
Leverage cryptographic primitives on existing public blockchains

asymmetric encryption

NIZK proofs
Data Privacy with NIZK Proofs

Alice’s data  0x3025...
Bob’s data    0x2543...

42
Data Privacy with NIZK Proofs

Alice’s data: 0x3025...
Bob’s data: 0x2543...

“increment my number by 1”

42
Data Privacy with NIZK Proofs

Alice’s data 0x3025...
Bob’s data 0x2543...

Transaction

Alice

"increment my number by 1"

0x0932...

42

Alice’s data 0x0932...
Bob’s data 0x2543...

43
Data Privacy with NIZK Proofs

Alice’s data: 0x3025...
Bob’s data: 0x2543...

42

“increment my number by 1”

0x0932...
transaction

43

Alice’s data: 0x0932...
Bob’s data: 0x2543...

NIZK proof

“the new value is correct”
Data Privacy with NIZK Proofs

Alice's data: 0x3025...
Bob's data: 0x2543...

Alice

"increment my number by 1"

42

Alice's data: 0x0932...
Bob's data: 0x2543...

0x0932...

transaction

NIZK proof

verified by miners

43

"the new value is correct"
Data Privacy with NIZK Proofs

Alice

"increment my number by 1"

42

Alice’s data
Bob’s data

0x3025...
0x2543...

transaction

0x0932...

NIZK proof

verified by miners

43

Alice’s data
Bob’s data

0x0932...
0x2543...

“the new value is correct”
Challenges

Incompleteness of NIZK Proofs
Challenges

Incompleteness of NIZK Proofs

e.g., can only incorporate fixed-size state

"Puzzle" by jeff from thenounproject.com
Challenges

Incompleteness of NIZK Proofs

e.g., can only incorporate fixed-size state

Dynamic arrays?

Entry *lookup* in contract

Entry *decryption* in proof
Challenges

Incompleteness of NIZK Proofs

- e.g., can only incorporate fixed-size state

- scattered logic

Dynamic arrays?

- Entry *lookup* in contract
- Entry *decryption* in proof

"Puzzle" by jeff from thenounproject.com
Challenges

Incompleteness of NIZK Proofs

Obfuscated Logic

E.g., can only incorporate fixed-size state

Dynamic arrays?

Entry *lookup* in contract

Entry *decryption* in proof

Scattered logic

"Puzzle" by jeff from thenounproject.com
Challenges

Knowledge Restrictions

Alice: “increase Bob’s value by 1”
Challenges

Knowledge Restrictions

Alice: “increase Bob’s value by 1”

can not generate proof
Challenges

Obfuscated Information Leaks

Alice: “overwrite Bob’s value with my value”
Challenges

Obfuscated Information Leaks

Alice: “overwrite Bob’s value with my value”

did Alice really want this?

"leak" by Quentin B. from thenounproject.com
Challenges

Incompleteness of NIZK Proofs

Knowledge Restrictions

Obfuscated Logic

Obfuscated Information Leaks
Challenges

- Incompleteness of NIZK Proofs
- Knowledge Restrictions
- Obfuscated Logic
- Obfuscated Information Leaks

This work: zkay
zkay Overview

zkay contract
zkay Overview

zkay contract
zkay Overview

zkay contract
zkay Overview

zkay contract

compilation

smart contract for public blockchain
zkay **Overview**

zkay contract → compilation → smart contract for public blockchain
zkay Overview

zkay contract → compilation → smart contract for public blockchain
zkay **Overview**

zkay contract

+ type system

**Compilation**

smart contract for public blockchain
contract MedStats {
    address hospital;
    uint count;
    mapping(address => bool) risk;

    constructor() {
        hospital = msg.sender; count = 0;
    }

    function record(address don, bool r) {
        require(hospital == msg.sender);
        risk[don] = r;
        count = count + (r ? 1 : 0);
    }
}
Example

contract MedStats {
    address hospital;
    uint count;
    mapping(address => bool) risk;

    constructor() {
        hospital = msg.sender; count = 0;
    }

    function record(address don, bool r) {
        require(hospital == msg.sender);
        risk[don] = r;
        count = count + (r ? 1 : 0);
    }
}

Medical Statistics

risk

count: 2
Example

```solidity
contract MedStats {
    address hospital;
    uint count;
    mapping(address => bool) risk;

    constructor() {
        hospital = msg.sender; count = 0;
    }

    function record(address don, bool r) {
        require(hospital == msg.sender);
        risk[don] = r;
        count = count + (r ? 1 : 0);
    }
}
```

Medical Statistics

- Risk: ! !
- Count: 2
contract MedStats {
    address hospital;
    uint count;
    mapping(address => bool) risk;

    constructor() {
        hospital = msg.sender; count = 0;
    }

    function record(address don, bool r) {
        require(hospital == msg.sender);
        risk[don] = r;
        count = count + (r ? 1 : 0);
    }
}
Example

```solidity
contract MedStats {
    address hospital;
    uint count;
    mapping(address => bool) risk;

    constructor() {
        hospital = msg.sender; count = 0;
    }

    function record(address don, bool r) {
        require(hospital == msg.sender);
        risk[don] = r;
        count = count + (r ? 1 : 0);
    }
}
```
Example

```solidity
contract MedStats {
    address hospital;
    uint count;
    mapping(address => bool) risk;

    constructor() {
        hospital = msg.sender; count = 0;
    }

    function record(address don, bool r) {
        require(hospital == msg.sender);
        risk[don] = r;
        count = count + (r ? 1 : 0);
    }
}
```
Example

```
contract MedStats {
    address hospital;
    uint count;
    mapping(address => bool) risk;

    constructor() {
        hospital = msg.sender; count = 0;
    }

    function record(address don, bool r) {
        require(hospital == msg.sender);
        risk[don] = r;
        count = count + (r ? 1 : 0);
    }
}
```

Record data of donor
Example

```
contract MedStats {
    address hospital;
    uint count;
    mapping(address => bool) risk;

    constructor() {
        hospital = msg.sender; count = 0;
    }

    function record(address don, bool r) {
        require(hospital == msg.sender);
        risk[don] = r;
        count = count + (r ? 1 : 0);
    }
}
```
contract MedStats {
    address hospital;
    uint count;
    mapping(address => bool) risk;

    constructor() {
        hospital = msg.sender; count = 0;
    }

    function record(address don, bool r) {
        require(hospital == msg.sender);
        risk[don] = r;
        count = count + (r ? 1 : 0);
    }
}
Idea: `datatype@owner`
zkay Privacy Annotations

```solidity
contract MedStats {
    final address hospital;
    uint@hospital count;
    mapping(address!x => bool@x) risk;

    constructor() {
        hospital = me; count = 0;
    }

    function record(address don, bool@me r) {
        require(hospital == me);
        risk[don] = reveal(r, don);
        count = count + (r ? 1 : 0);
    }
}
```

- count is *private to* hospital
- hospital is the *owner* of count
zkay Privacy Annotations

contract MedStats {
    final address hospital;
    uint@hospital count;
    mapping(address!x => bool@x) risk;

    constructor() {
        hospital = me; count = 0;
    }

    function record(address don, bool@me r) {
        require(hospital == me);
        risk[don] = reveal(r, don);
        count = count + (r ? 1 : 0);
    }
}

count is private to hospital
hospital is the owner of count
risk[a] is private to address a
zkay **Privacy Annotations**

```solidity
class MedStats {
    final address hospital;
    uint@hospital count;
    mapping(address => bool) risk;

    constructor() {
        hospital = me; count = 0;
    }

    function record(address don, bool me r) {
        require(hospital == me);
        risk[don] = reveal(r, don);
        count = count + (r ? 1 : 0);
    }
}
```

- **count** is *private to hospital*
- **hospital** is the *owner* of **count**
- **risk[a]** is private to address **a**
- **r** is private to the caller (**@me**)
zkay Privacy Annotations

contract MedStats {
    final address hospital;
    uint@hospital count;
    mapping(address!x => bool@x) risk;

    constructor() {
        hospital = me; count = 0;
    }

    function record(address don, bool@me r) {
        require(hospital == me);
        risk[don] = reveal(r, don);
        count = count + (r ? 1 : 0);
    }
}

count is private to hospital
hospital is the owner of count

risk[a] is private to address a

r is private to the caller (@me)

don is public (@all)
contract MedStats {
    final address hospital;
    uint@hospital count;
    mapping(address!x => bool@x) risk;

    constructor() {
        hospital = me; count = 0;
    }

    function record(address don, bool@me r) {
        require(hospital == me);
        risk[don] = reveal(r, don);
        count = count + (r ? 1 : 0);
    }
}
zkay Privacy Annotations

```solidity
contract MedStats {
    final address hospital;
    uint@hospital count;
    mapping(address!x => bool@x) risk;

    constructor() {
        hospital = me; count = 0;
    }

    function record(address don, bool@me r) {
        require(hospital == me);
        risk[don] = reveal(r, don);
        count = count + (r ? 1 : 0);
    }
}
```

Logic *not obfuscated*
Type System Goals

No unintended information leaks
Type System Goals

No unintended information leaks

Realizability using NIZK proofs
contract MedStats {
    final address hospital;
    uint@hospital count;
    mapping(address!x => bool@x) risk;

    constructor() {
        hospital = me; count = 0;
    }

    function record(address don, bool@me r) {
        require(hospital == me);
        risk[don] = reveal(r, don);
        count = count + (r ? 1 : 0);
    }
}
Preventing Unintended Leaks

contract MedStats {
    final address hospital;
    uint@hospital count;
    mapping(address!x => bool@x) risk;

    constructor() {
        hospital = me; count = 0;
    }

    function record(address don, bool@me r) {
        require(hospital == me);
        risk[don] = reveal(r, don);
        count = count + (r ? 1 : 0);
    }
}
contract MedStats {
    final address hospital;
    uint@hospital count;
    mapping(address!x => bool@x) risk;

    constructor() {
        hospital = me; count = 0;
    }

    function record(address don, bool@me r) {
        require(hospital == me);
        risk[don] = r;
        count = count + (r ? 1 : 0);
    }
}
Preventing Unintended Leaks

contract MedStats {
    final address hospital;
    uint@hospital count;
    mapping(address!x => bool@x) risk;

    constructor() {
        hospital = me; count = 0;
    }

    function record(address don, bool@me r) {
        require(hospital == me);
        risk[don] = r;
        count = count + (r ? 1 : 0);
    }
}

Disallow implicit re-/declassification
Preventing Unintended Leaks

contract MedStats {
  final address hospital;
  uint@hospital count;
  mapping(address!x => bool@x) risk;

  constructor() {
    hospital = me; count = 0;
  }

  function record(address don, bool@me r) {
    require(hospital == me);
    risk[don] = reveal(r, don);
    count = count + (r ? 1 : 0);
  }
}
Preventing Unintended Leaks

contract MedStats {
    final address hospital;
    uint@hospital count;
    mapping(address!x => bool@x) risk;

    constructor() {
        hospital = me; count = 0;
    }

    function record(address don, bool@me r) {
        require(hospital == me);
        risk[don] = reveal(r, don);
        count = count + (r ? 1 : 0);
    }
}
Preventing Unintended Leaks

contract MedStats {
    final address hospital;
    uint@hospital count;
    mapping(address!x => bool@x) risk;

    constructor() {
        hospital = me; count = 0;
    }

    function record(address don, bool@me r) {
        require(hospital == me);
        risk[don] = reveal(r, don);
        count = count + (r ? 1 : 0);
    }
}

Type of compound expressions is conservative
Ensuring Realizability

Alice

data[bob] = data[bob] + 1;
Ensuring Realizability

Alice

data[bob] = data[bob] + 1;

@bob
Ensuring Realizability

Alice

```plaintext
data[bob] = data[bob] + 1;
```

@bob can only read @me or @all

29
Ensuring Realizability

```solidity
contract MedStats {
    final address hospital;
    uint@hospital count;
    mapping(address!x => bool@x) risk;

    constructor() {
        hospital = me; count = 0;
    }

    function record(address don, bool@me r) {
        require(hospital == me);
        risk[don] = reveal(r, don);
        count = count + (r ? 1 : 0);
    }
}
```

Can only read self-owned or public variables
Ensuring Realizability

```solidity
contract MedStats {
    final address hospital;
    uint@hospital count;
    mapping(address!x => bool@x) risk;

    constructor() {
        hospital = me; count = 0;
    }

    function record(address don, bool@me r) {
        require(hospital == me);
        risk[don] = reveal(r, don);
        count = count + (r ? 1 : 0);
    }
}
```

Can only read self-owned or public variables
Ensuring Realizability

contract MedStats {
    final address hospital;
    uint@hospital count;
    mapping(address!x => bool@x) risk;

    constructor() {
        hospital = me; count = 0;
    }

    function record(address don, bool@me r) {
        require(hospital == me);
        risk[don] = reveal(r, don);
        count = count + (r ? 1 : 0);
    }
}

Sometimes, static analysis is required
Ensuring Realizability

contract MedStats {
    final address hospital;
    uint@hospital count;
    mapping(address!x => bool@x) risk;

    constructor() {
        hospital = me; count = 0;
    }

    function record(address don, bool@me r) {
        require(hospital == me);
        risk[don] = reveal(r, don);
        count = count + (r ? 1 : 0);
    }
}
Compilation
function record(bool@me r) {
    require(hospital == me);
    count = count + (r ? 1 : 0);
}
function record(bool@me r) {
    require(hospital == me);
    count = count + (r ? 1 : 0);
}

function record(bin r, bin z, bin p) {
    require(hospital == me);
    bin old = count; count = z;
    verify(p, z, r, old, pk(me));
}
function record(bool@me r) {
    require(hospital == me);
    count = count + (r ? 1 : 0);
}

function record(bin r, bin z, bin p) {
    require(hospital == me);
    bin old = count; count = z;
    verify(p, z, r, old, pk(me));
}
function record(bool@me r) {
    require(hospital == me);
    count = count + (r ? 1 : 0);
}

function record(bin r, bin z, bin p) {
    require(hospital == me);
    bin old = count; count = z;
    verify(p, z, r, old, pk(me));
}
Compilation Example

```javascript
function record(bool@me r) {
    require(hospital == me);
    count = count + (r ? 1 : 0);
}

function record(bin r, bin z, bin p) {
    require(hospital == me);
    bin old = count; count = z;
    verify(p, z, r, old, pk(me));
}
```
Compilation Example

function record(bool@me r) {
    require(hospital == me);
    count = count + (r ? 1 : 0);
}

function record(bin r, bin z, bin p) {
    require(hospital == me);
    bin old = count; count = z;
    verify(p, z, r, old, pk(me));
}
function record(bool@me r) {
    require(hospital == me);
    count = count + (r ? 1 : 0);
}

function record(bin r, bin z, bin p) {
    require(hospital == me);
    bin old = count; count = z;
    verify(p, z, r, old, pk(me));
}
I know $sk$ such that

$$old + (r ? 1 : 0)$$

**Compilation Example**

```plaintext
function record(bool@me r) {
  require(hospital == me);
  count = count + (r ? 1 : 0);
}
```

```plaintext
private function record(bin r, bin z, bin p) {
  require(hospital == me);
  bin old = count; count = z;
  verify(p, z, r, old, pk(me));
}
```

```plaintext
public function record(bool@me r) {
  require(hospital == me);
}
```
I know sk such that

\[
\text{old} + (\text{r} ? 1 : 0) \\
\text{dec(old, sk)} + (\text{dec(r, sk)} ? 1 : 0)
\]
I know \( sk \) such that

\[
    \text{old} + (r ? 1 : 0) \\
    \text{dec(old, sk)} + (\text{dec(r, sk)} ? 1 : 0) \\
    z == \text{enc(dec(old, sk)} + (\text{dec(r, sk)? 1 : 0), pk(me))
\]
Compilation

Figure 5: Overview of zkay transformations. We write $e$ at $a$ to indicate that $a$ has privacy type $e$. The symbol $\alpha$ denotes a fresh variable.
Privacy Guarantees
Data Privacy

Ideal world

Input (specification)

Real world

Output (realization)
Data Privacy

ideal world

input (specification)

output (realization)

real world

Everything Eve can learn here...
Data Privacy

*ideal world*

input (specification)

→

output (realization)

*real world*

...she can also learn here...

Everything Eve can learn here...
Data Privacy

ideal world

input
(specification)

real world

output
(realization)

symbolic,
Dolev-Yao style

Everything Eve can
learn here...

...she can also learn here
Data Privacy

**ideal world**

input (specification)

---

output (realization)

symbolic, Dolev-Yao style

Everything Eve can learn here...

---

...she can also learn here

according to privacy types

---

real world
Data Privacy

Theorem: Compiled contracts are *private* w.r.t. their input contracts

...she can also learn here... according to privacy types

Everything Eve can learn here...
Implementation

zkay

Contract

eth-sri/zkay
Implementation

zkay

Contract

Type-check & Compilation

Proof Statements

eth-sri/zkay

SOL
Implementation

zkay

Contract

Type-check & Compilation

Proof Statements

ZoKrates

Eberhardt et al. IEEE Blockchain ‘18

eth-sri/zkay

SOL

verify
Implementation

zkay

Contract

Type-check & Compilation

Proof Statements

ZoKrates

Eberhardt et al. IEEE Blockchain ‘18

SOL

deploy

EVM

eth-sri/zkay

deploy

EVM

SOL

verify
Implementation

zkay

Contract

Type-check & Compilation

Proof Statements

ZoKrates

Eberhardt et al. IEEE Blockchain '18

SOL

deploy

EVM

Transformation

Transactions

SOL

verify

eth-sri/zkay
Evaluation

- Expressiveness?
- Gas costs?
Evaluation

10 Example Contracts

22 – 79 loc
Evaluation

10 Example Contracts

wide range of domains

- Insurance & Finance
- Healthcare
- Gambling
- ...
Evaluation

10 Example Contracts

22 – 79 loc

wide range of domains

Insurance & Finance

Healthcare

Gambling

Small example scenarios

4 – 9 transactions
Evaluation

10 Example Contracts
22 – 79 loc

Small example scenarios
4 – 9 transactions

Wide range of domains
Insurance & Finance
Healthcare
Gambling
...

zkay is expressive
On-Chain Costs

Private Transactions
On-Chain Costs

Private Transactions

dominated by constant-cost* NIZK proof verification

c.a. $10^6$ Gas per tx
On-Chain Costs

Private Transactions

c. $0.50 per tx (1 ETH = 170 US$)
c. $10^6$ Gas per tx

dominated by constant-cost* NIZK proof verification
On-Chain Costs

Private Transactions

dominated by constant-cost* NIZK proof verification

c. $0.50 (1 ETH = 170 US$)

c. $10^6$ Gas per tx

gas costs are moderate
Summary

zkay

data type @ owner

eth-sri/zkay

expressive

moderate costs
Icon Credit

Icons by FontAwesome (CC BY 4.0)

Pick icon made by Freepik from www.flaticon.com

"leak" by Quentin B. from thenounproject.com
"Puzzle" by jeff from thenounproject.com