zkay: **Specifying and Enforcing Data Privacy in Smart Contracts**

Samuel Steffen

Benjamin Bichsel

Mario Gersbach

Noa Melchior

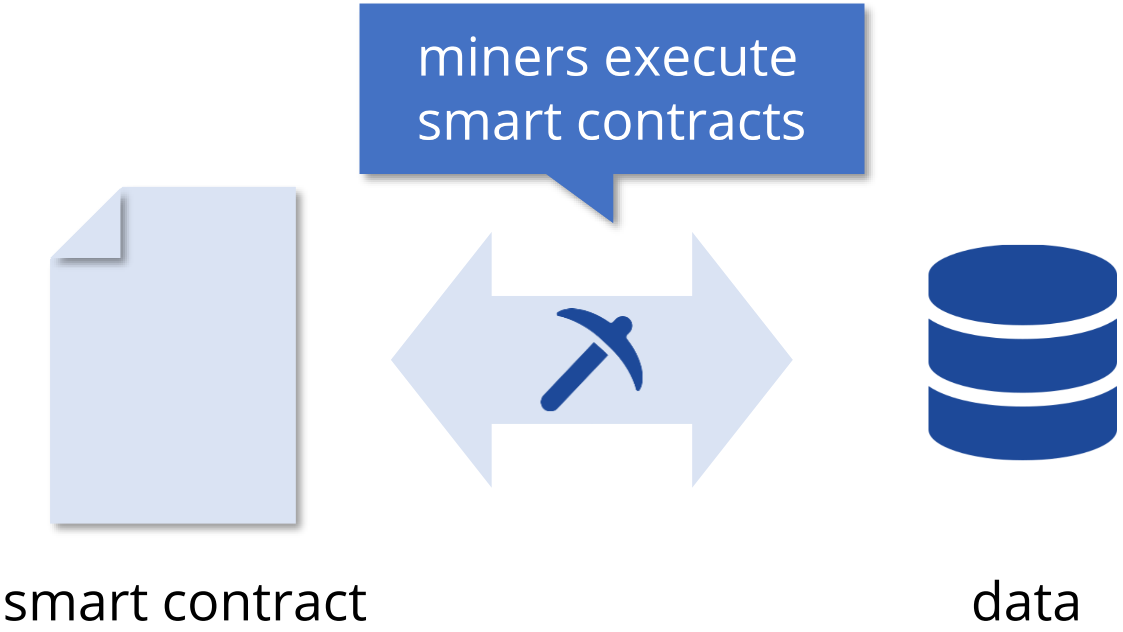
Petar Tsankov

Martin Vechev

ETH zürich

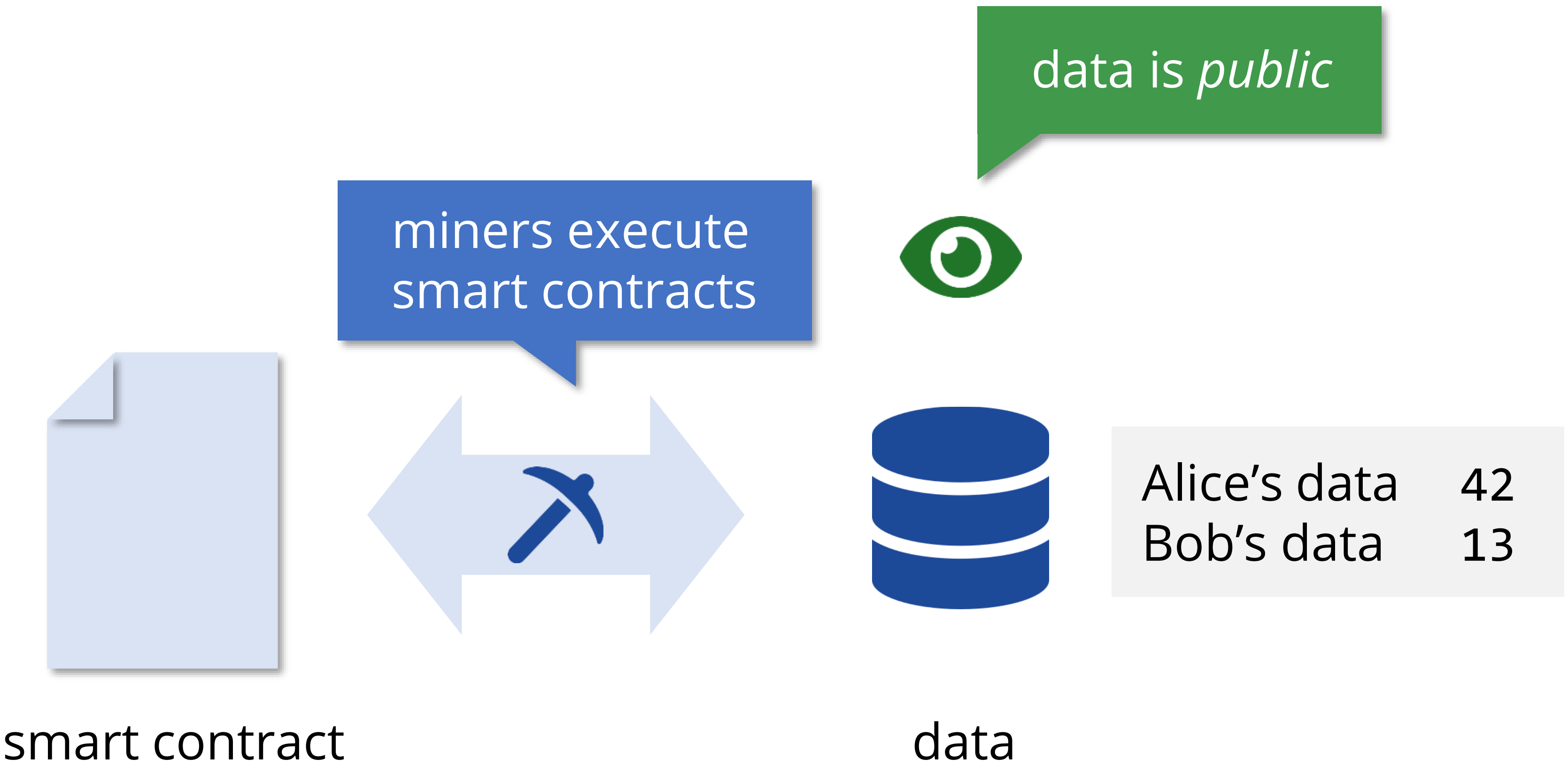
SRILAB

Smart Contracts



Pick icon made by Freepik from www.flaticon.com
Icons by FontAwesome (CC BY 4.0)

Smart Contracts



Pick icon made by Freepik from www.flaticon.com
Icons by FontAwesome (CC BY 4.0)

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?

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 16, 2017

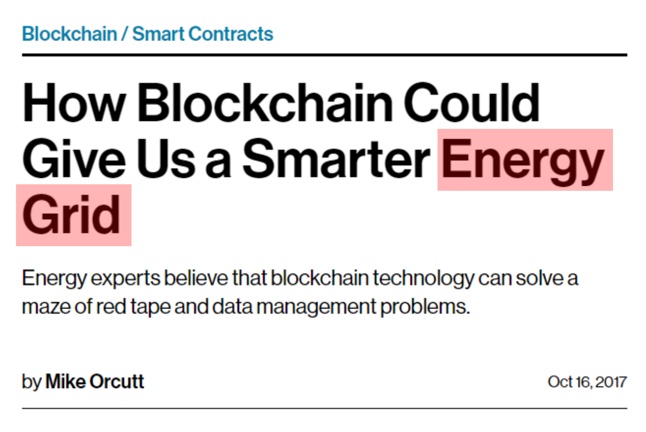
https://www.technologyreview.com/s/609077/how-blockchain-could-give-us-a-smarter-energy-grid/

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.

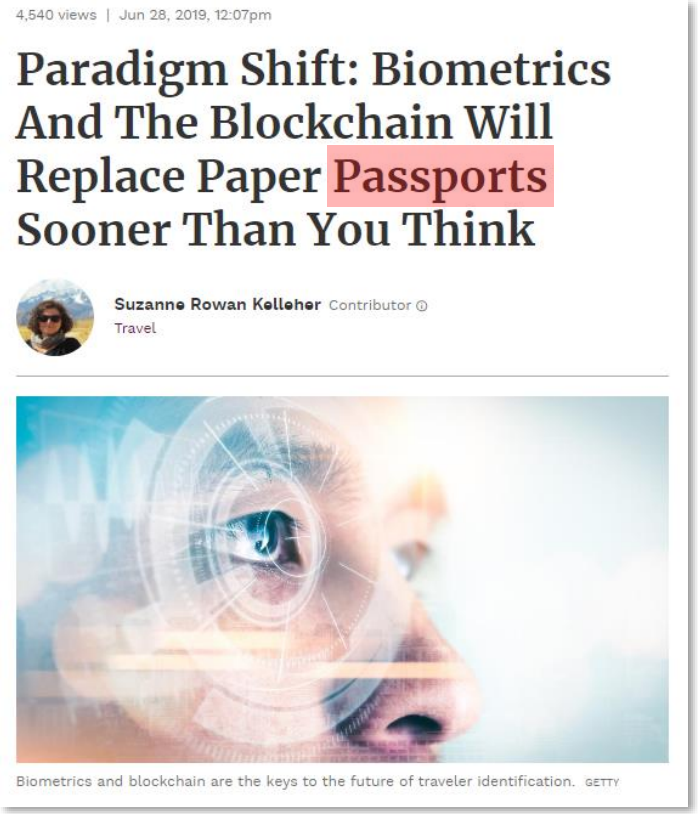
by Mike Orcutt Oct 16, 2017

https://www.technologyreview.com/s/609077/how-blockchain-could-give-us-a-smarter-energy-grid/


Medical data and the rise of blockchain

6th July 2018

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


4,540 views | Jun 28, 2019, 12:07pm

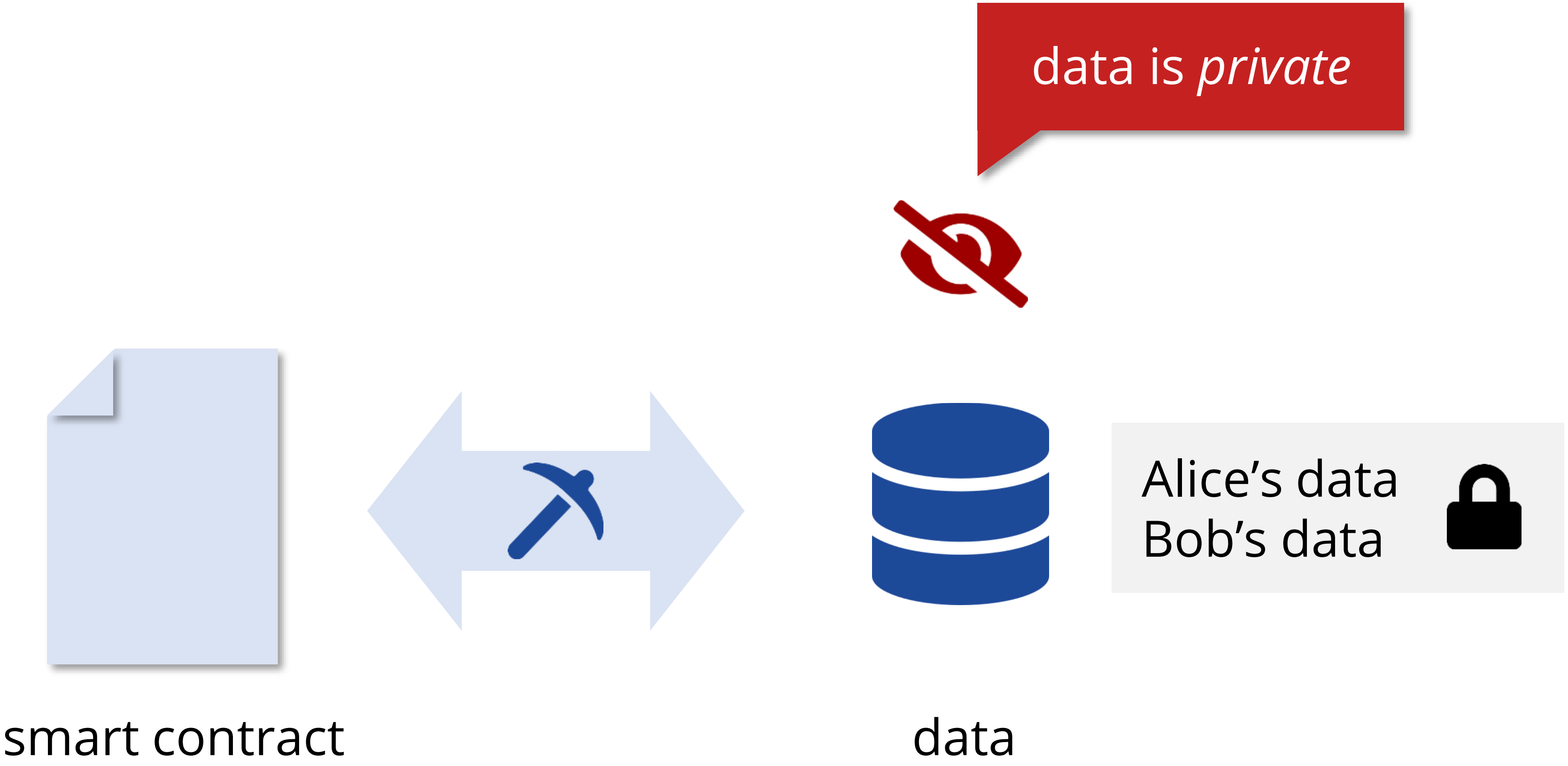
Paradigm Shift: Biometrics And The Blockchain Will Replace Paper Passports Sooner Than You Think

Suzanne Rowan Kelleher Contributor
Travel

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

https://www.forbes.com/sites/suzannerowankelleher/2019/06/28/paradigm-shift-biometrics-and-the-blockchain-will-replace-paper-passports-sooner-than-you-think/

Data Privacy for Smart Contracts



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]

only for payments

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

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

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

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

42

Alice’s data	`0x3025…`
Bob’s data	`0x2543…`

Data Privacy with NIZK Proofs

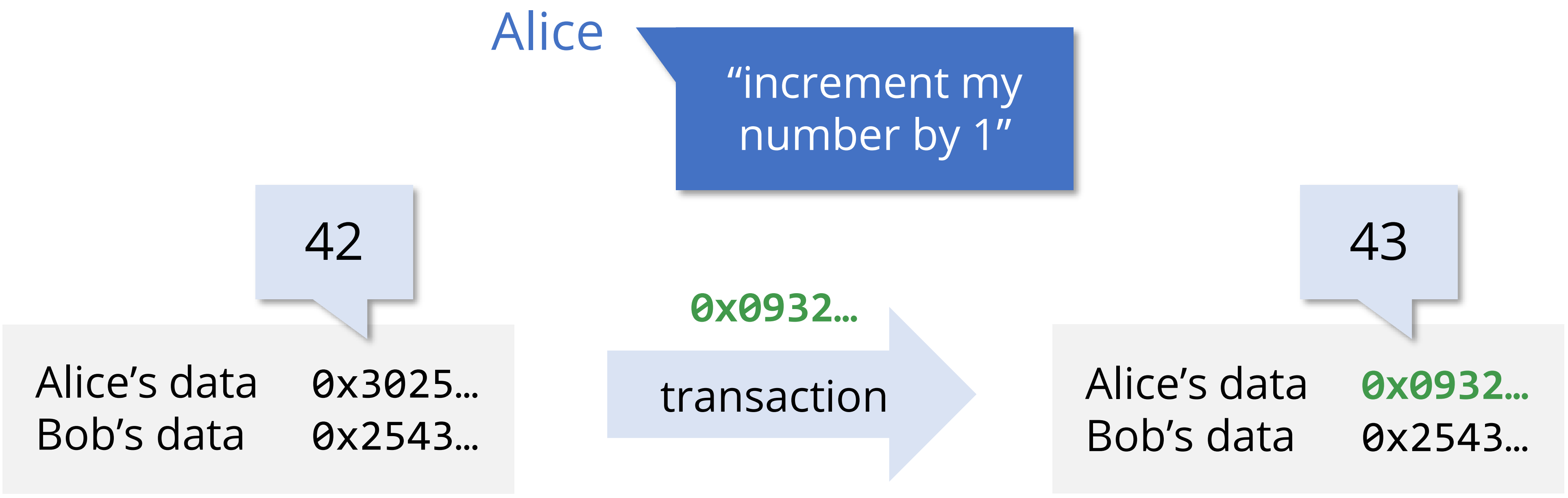
Alice

“increment my number by 1”

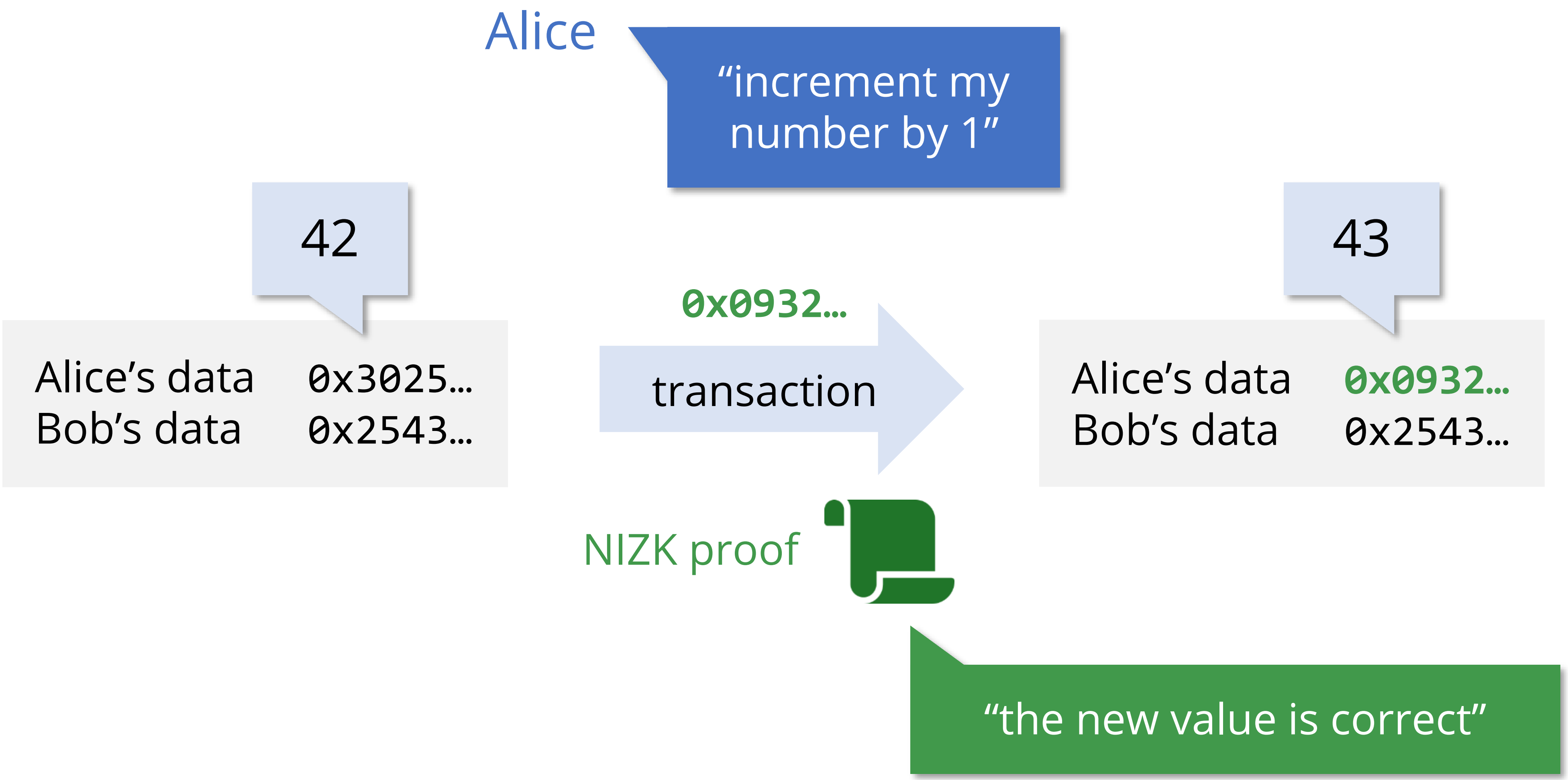
42

Alice’s data	`0x3025…`
Bob’s data	`0x2543…`

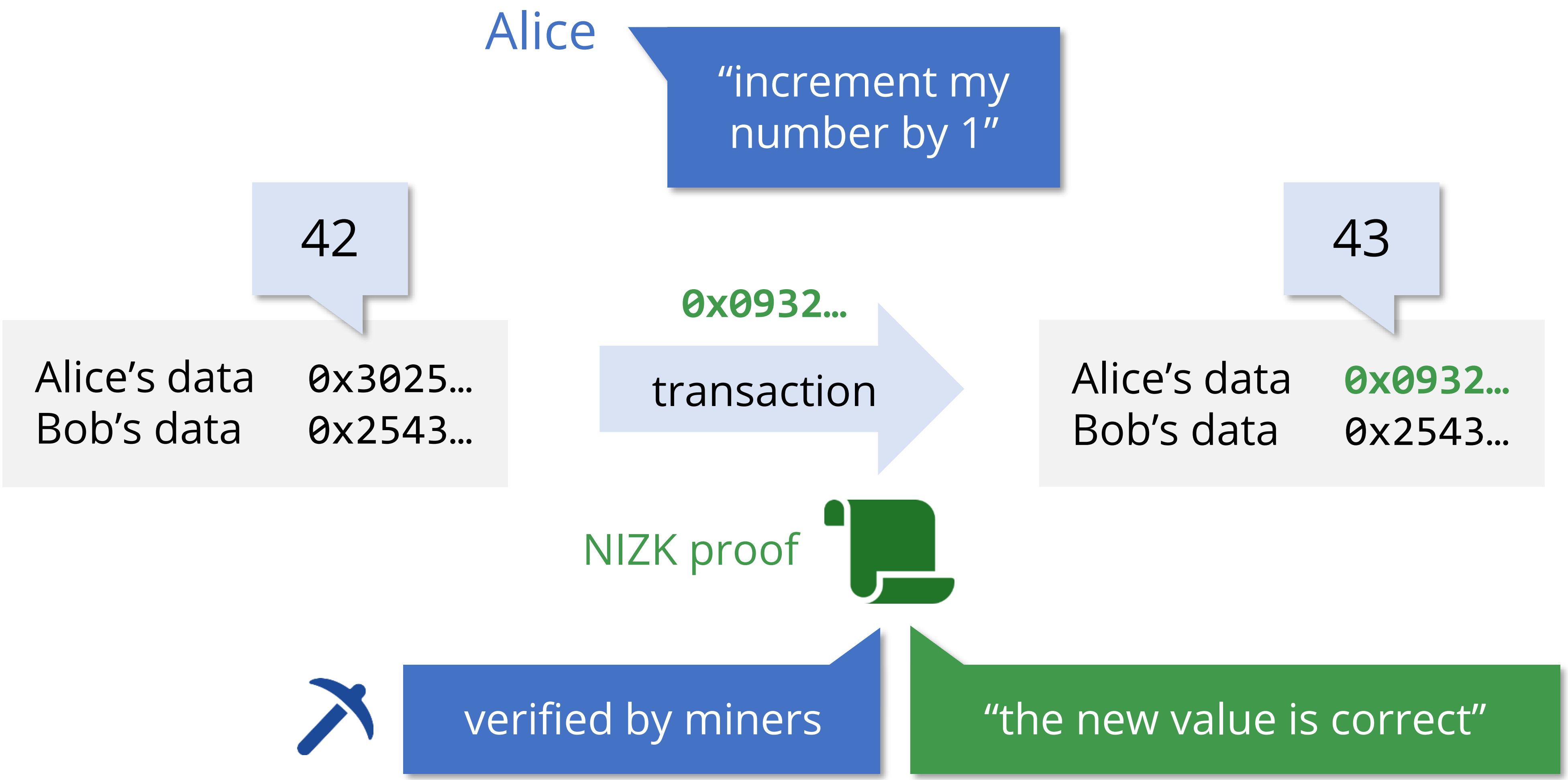
Data Privacy with NIZK Proofs



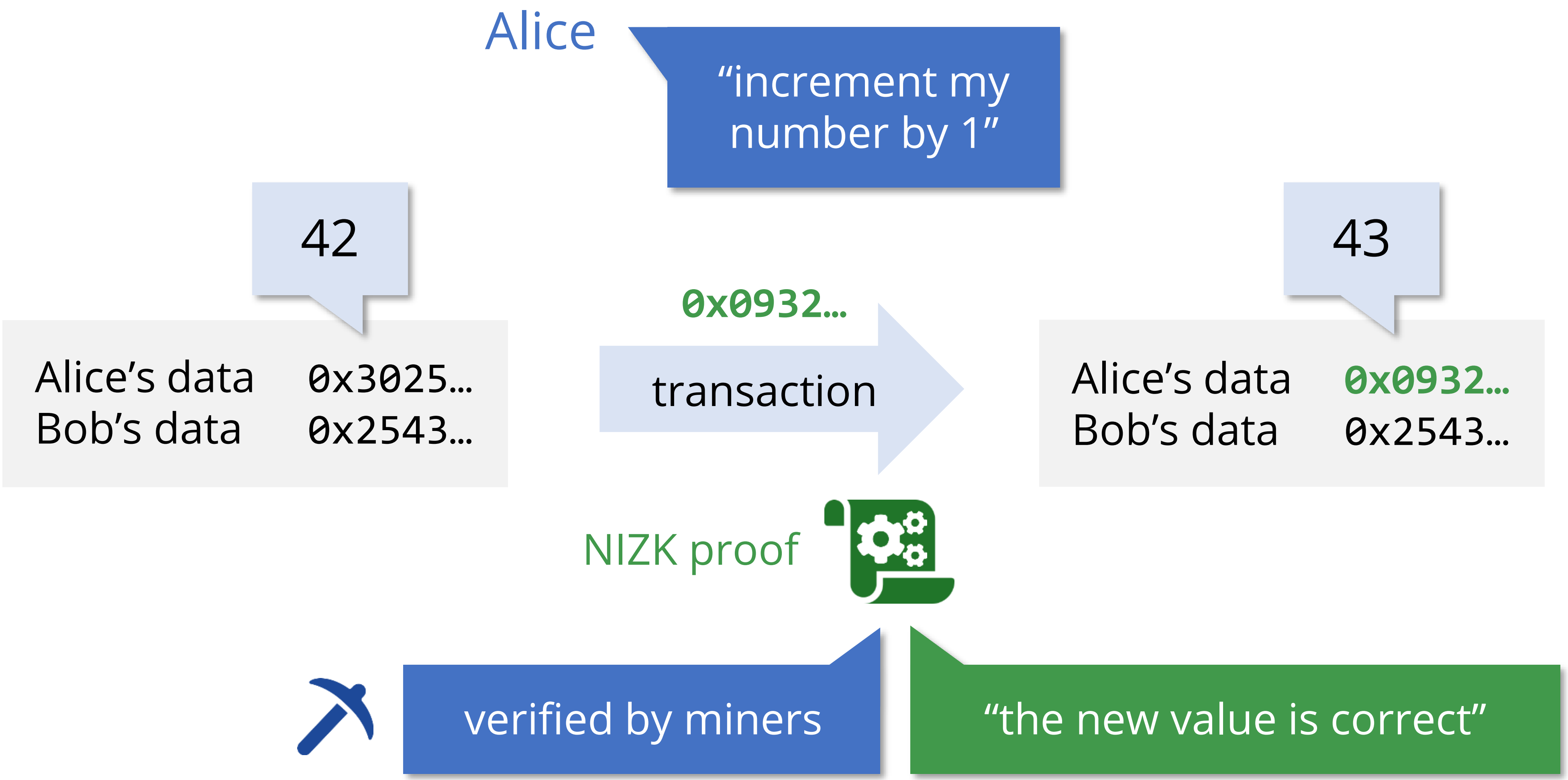
Data Privacy with NIZK Proofs



Data Privacy with NIZK Proofs

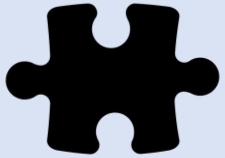


Data Privacy with NIZK Proofs



Challenges

Incompleteness of
NIZK Proofs

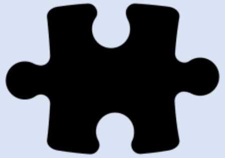


"Puzzle" by jeff from thenounproject.com

Challenges

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

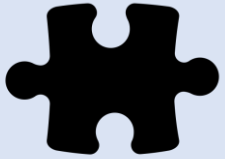
Incompleteness of NIZK Proofs



"Puzzle" by jeff from thenounproject.com

Challenges

Incompleteness of
NIZK Proofs



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

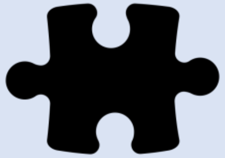
Dynamic arrays?

Entry *lookup*
in contract

Entry *decryption*
in proof

Challenges

Incompleteness of NIZK Proofs



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

Dynamic arrays?

Entry *lookup* in contract

Entry *decryption* in proof

scattered logic

"Puzzle" by jeff from thenounproject.com

Challenges

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

Incompleteness of NIZK Proofs

Dynamic arrays?

Entry *lookup* in contract

scattered logic

Entry *decryption* in proof

Obfuscated 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”

"leak" by Quentin B. from thenounproject.com

Challenges

Obfuscated Information Leaks 

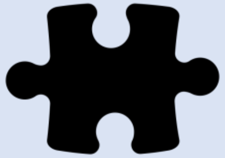
Alice: “overwrite Bob’s value with my value”

did Alice really want this?

"leak" by Quentin B. from thenounproject.com

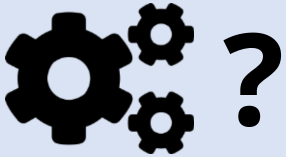
Challenges

Incompleteness of NIZK Proofs



Knowledge Restrictions

Obfuscated Logic



Obfuscated Information Leaks

Challenges

Incompleteness of NIZK Proofs

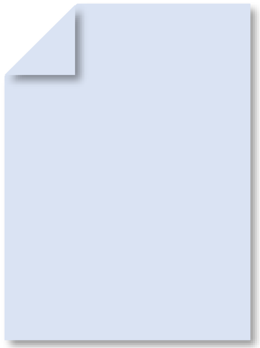
Knowledge Restrictions

This work: zkay

Obfuscated Logic ?

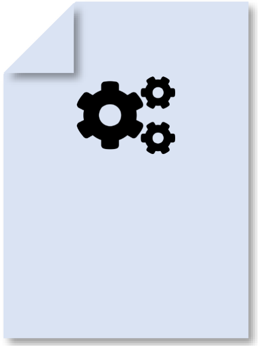
Obfuscated Information Leaks ?

zkay **Overview**



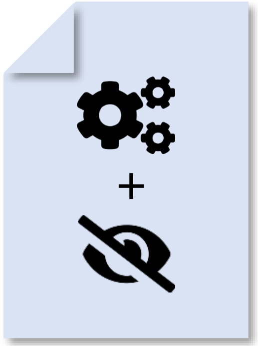
zkay contract

zkay **Overview**



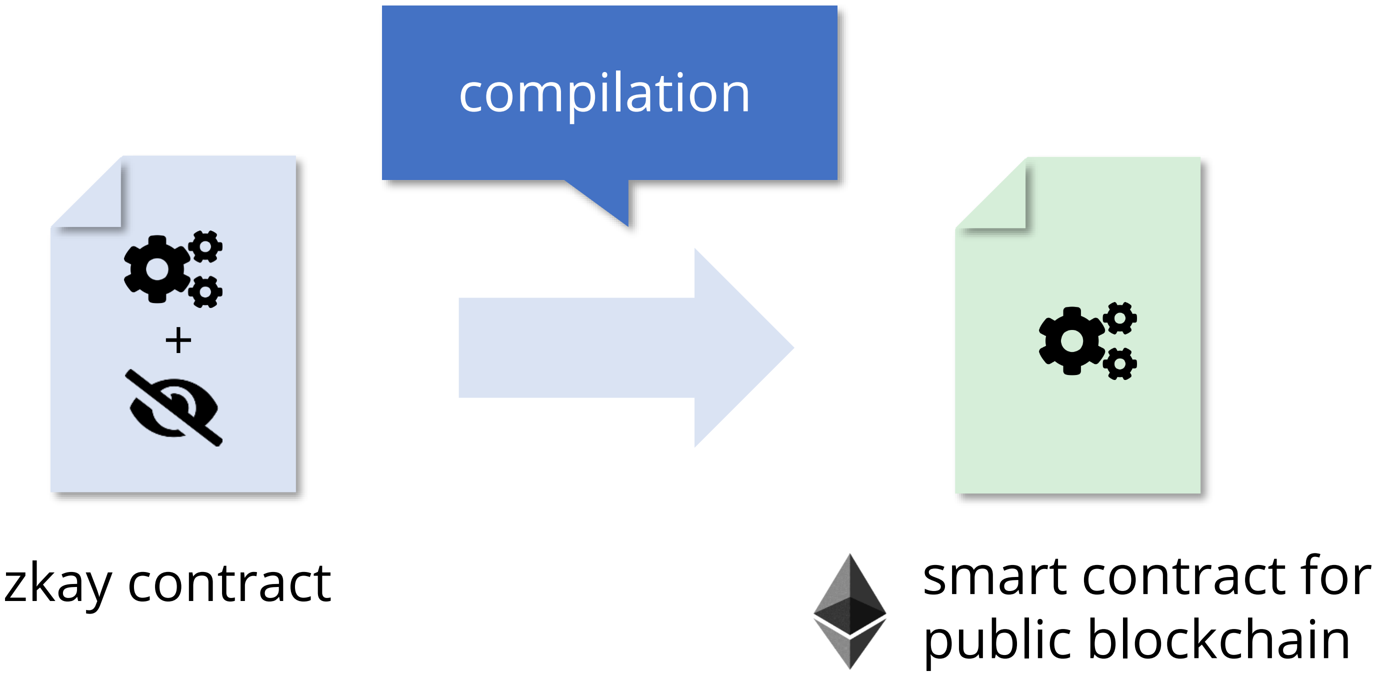
zkay contract

zkay **Overview**

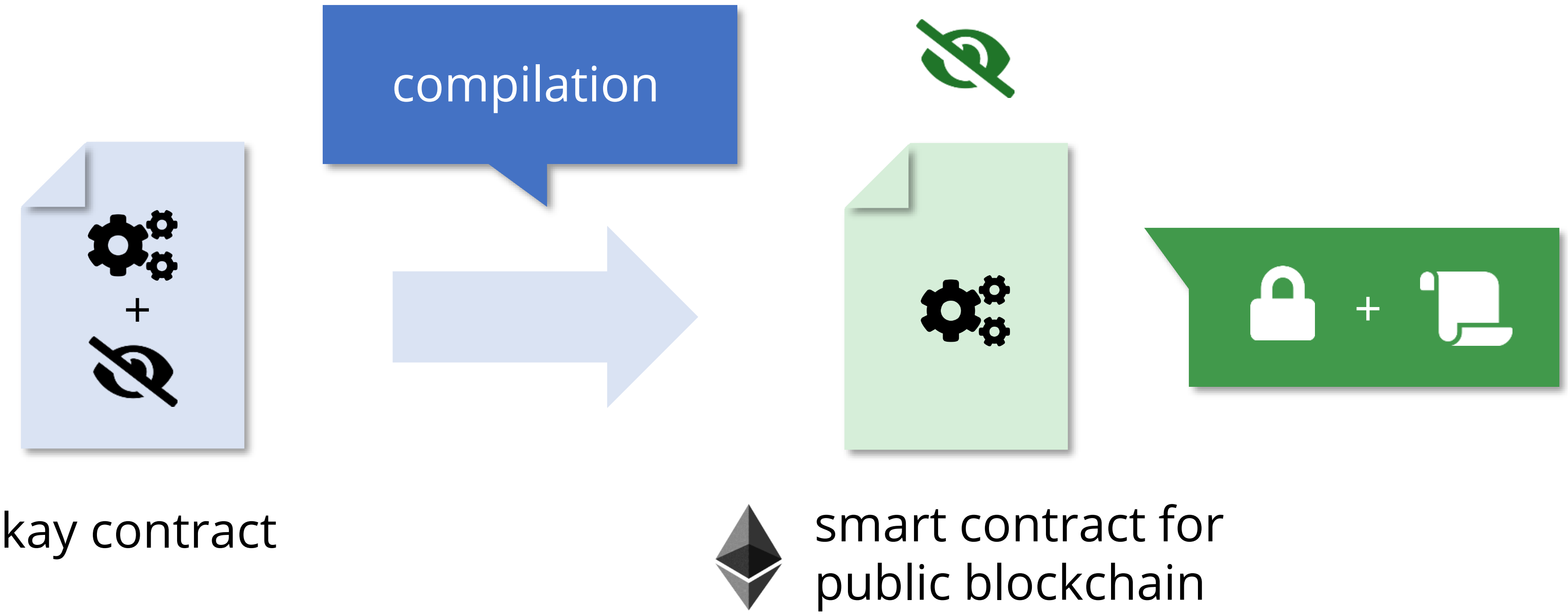


zkay contract

zkay **Overview**



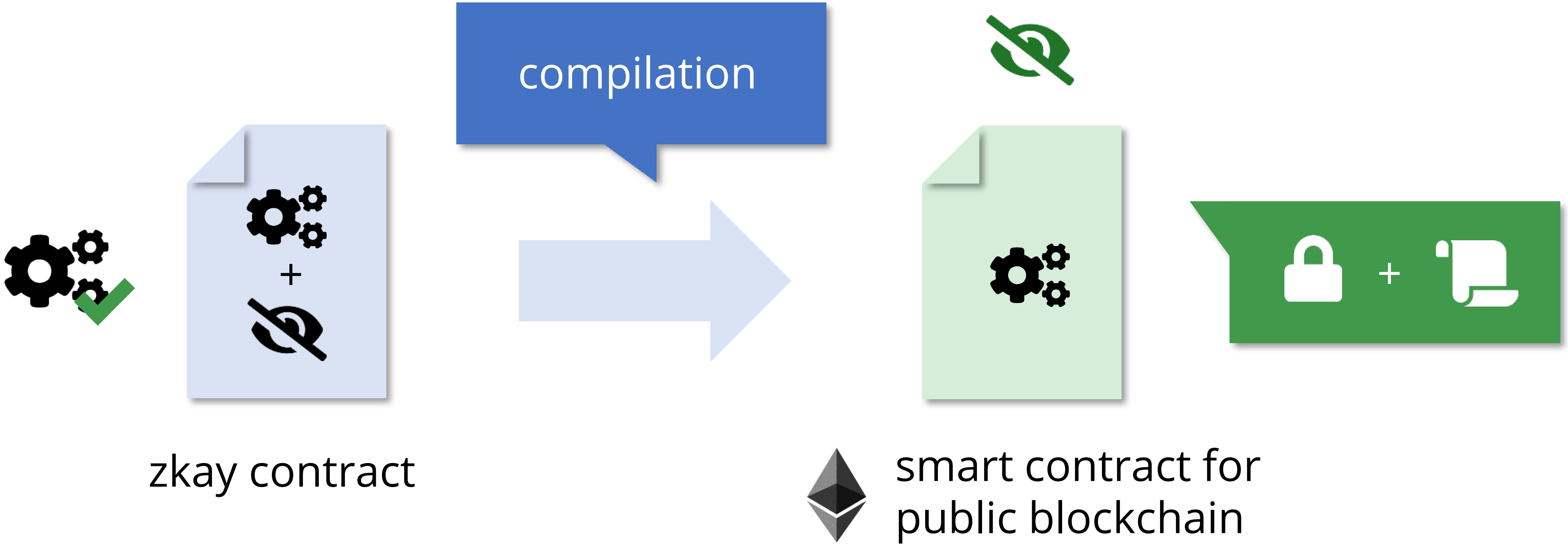
zkay **Overview**



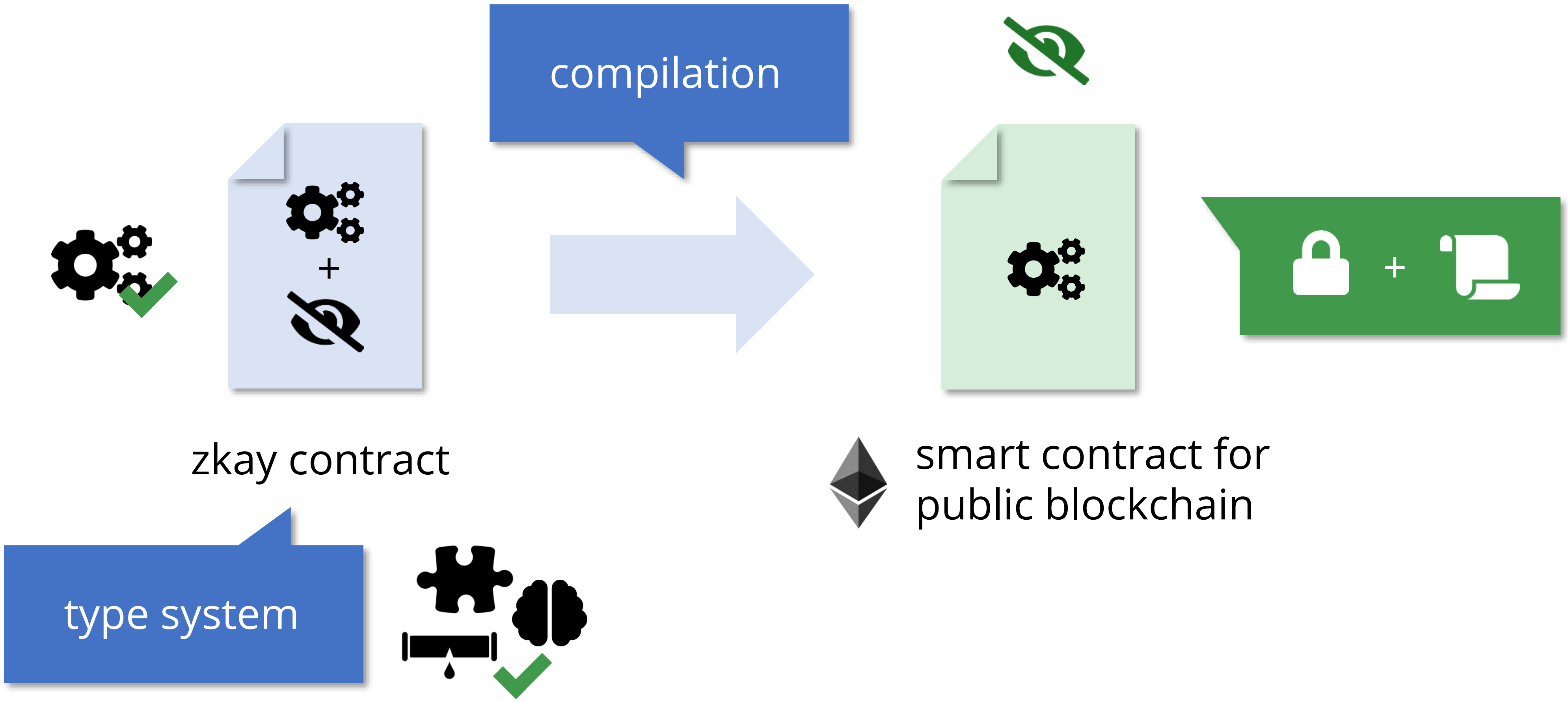
zkay contract

smart contract for
public blockchain

zkay **Overview**



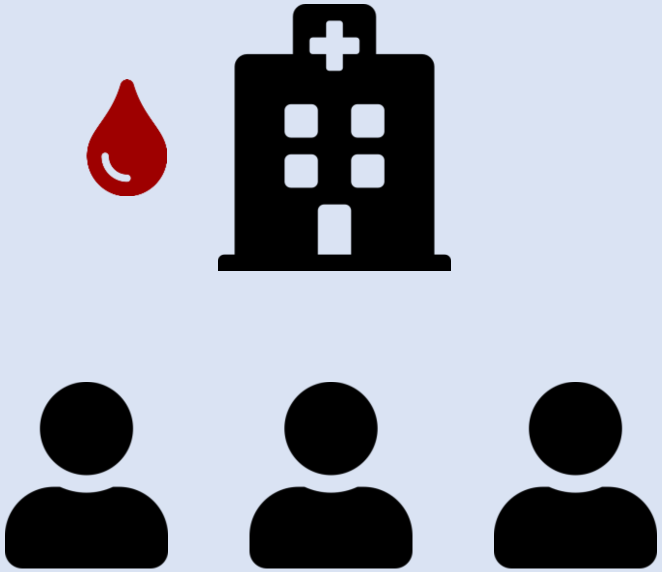
zkay **Overview**



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);
   }
}
```

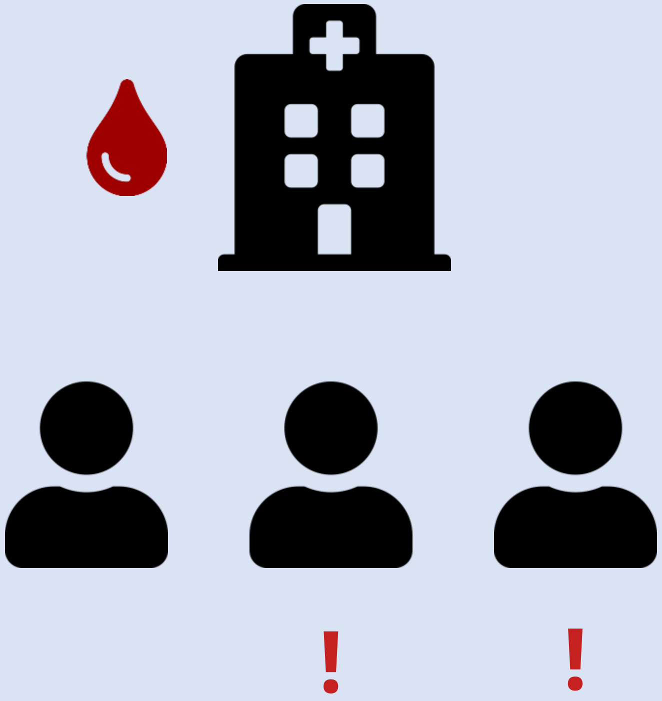
Medical Statistics



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

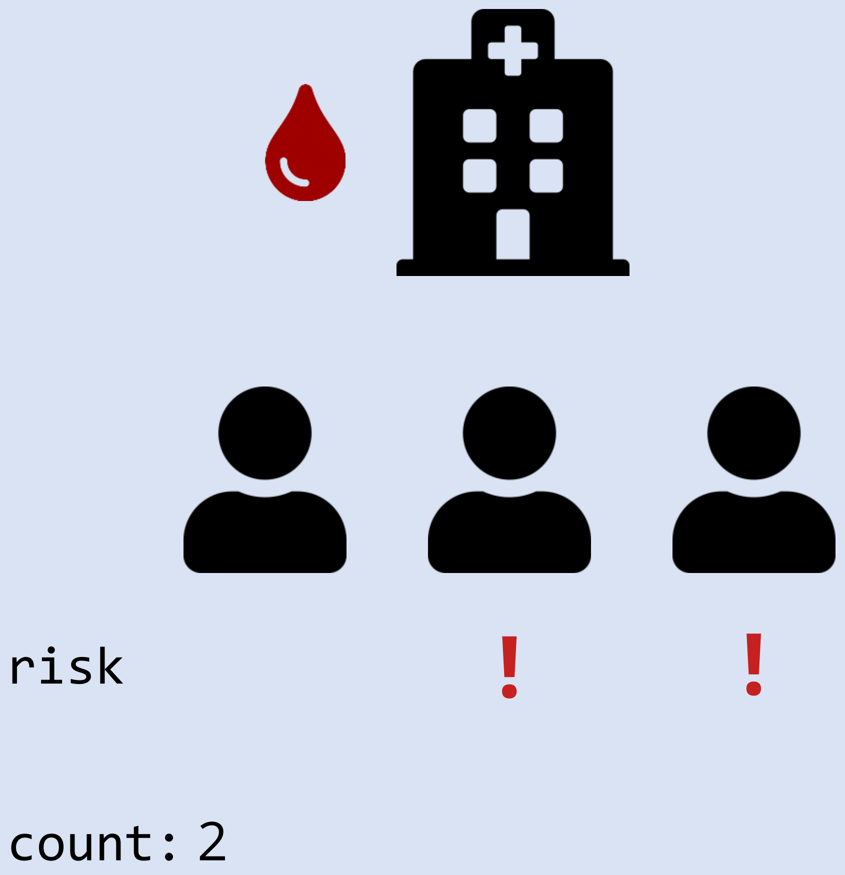
count: 2

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



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);
   }
}
```

Initialization

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);
   }
}
```

Record data of donor

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);
   }
}
```

Record data of donor

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);
  }
}
```

Record data of donor

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);
   }
}
```

Record data of donor

Privacy?

```
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);
   }
}
```

Sensitive information must remain *private*

zkay **Privacy Annotations**

Idea:

```
datatype@owner
```

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

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**

```
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)

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`)

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**)

reclassify r for don

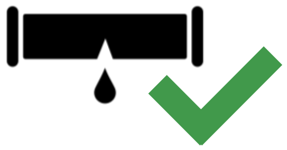
zkay **Privacy Annotations**

```
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);
   }
}
```

Logic *not obfuscated*

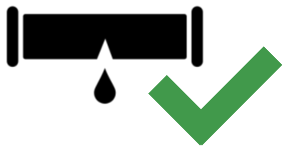
Type System Goals

No unintended information leaks

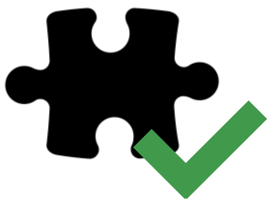


Type System Goals

No unintended information leaks



Realizability using NIZK proofs



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);
   }
}
```

@me

@all

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);
   }
}
```

OK

@me

@all

Allow implicit *classification*

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);

   }
}
```

@don

@me

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 =
      risk[don] = r;
      count = count + (r ? 1 : 0);
   }
}
```

disallow

@don

@me

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
  }
}
```

OK

requires **reveal**

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);
   }
}
```

@me

@all

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);
  }
}
```

@me

@me

@all

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

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

@bob

can only read **`@me`** or **`@all`**

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);
@hospital
}
```

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(hospita
    risk[don] = re          );
    count = count + (r ? 1 : 0);

}
```

?

@hospital

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);
```

`@hospital`

```
}
```

can prove `hospital == me`

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);
   }
}
```

can prove `hospital == me`

@me

OK

Sometimes, *static analysis* is required

Compilation

Compilation Example

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

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));
}
```

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));
}
```

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));
}
```

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));
}
```

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));
}
```

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));
}
```

public

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));
}
```

public

private

I know sk such that

old + (r ? 1 : 0)

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));
}
```

public

private

I know sk such that

```
      old     + (     r       ? 1 : 0)
dec(old, sk) + (dec(r, sk) ? 1 : 0)
```

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));
}
```

public

private

I know sk such that

```
                old      + (    r        ? 1 : 0)
        dec(old, sk) + (dec(r, sk) ? 1 : 0)
z == enc(dec(old, sk) + (dec(r, sk) ? 1 : 0), pk(me))
```

Compilation

```
function f(...) {
  P;
  return e;
}
```

```
function f̄(..., 𝓕, bin proof) {
  T(P)
  verify_ϕ(proof, 𝓟);
  return T_e(e);
}
```

for ϕ: $(\mathcal{P}, \mathcal{S}) \to \{0, 1\}$

(a) Transformation of a zkay function.

$$out(e, \alpha) ::= v_i \qquad \text{(invariant: } e@\texttt{me} \text{ or } e@\texttt{all})$$

$$\mathcal{F} \leftarrow \mathcal{F}, v_i$$

$$\mathcal{P} \leftarrow \mathcal{P}, v_i, \texttt{pk}(\alpha)$$

$$\mathcal{S} \leftarrow \mathcal{S}, R_i$$

$$\phi \leftarrow \phi;\ v_i == \texttt{enc}(\ T_\phi(e)\ , R_i, \texttt{pk}(\alpha))\ ;$$

(b) Transformation $out(e, \alpha)$. If $\alpha = \texttt{all}$, the highlighted part is omitted.

$$in(e, \alpha) ::= \texttt{dec}_\tau(\ v_i,\ \texttt{sk}) \qquad \text{(invariant: } e@\alpha, \alpha \in \{\texttt{me}, \texttt{all}\})$$

$$\text{add to } T(P) \quad : \quad \tau'@\texttt{all}\ \ v_i = T_e(e); \qquad \text{for } \tau' ::= \begin{cases} \tau & \alpha = \texttt{all} \\ \texttt{bin} & \alpha \neq \texttt{all} \end{cases}$$

$$\mathcal{P} \leftarrow \mathcal{P}, v_i$$

$$\mathcal{S} \leftarrow \mathcal{S}, \texttt{sk}$$

(c) Transformation $in(e, \alpha)$. If $\alpha = \texttt{all}$, the highlighted part is omitted.

$$T(P_1; P_2) ::= T(P_1); T(P_2) \tag{1}$$

$$T(L@\alpha = e@\alpha) ::= T_L(L) = T_e(e) \qquad \text{(we use } T_L = T_e) \tag{2}$$

$$T(L@\alpha = e@\texttt{all}) ::= T_L(L) = out(e, \alpha) \qquad (\alpha \neq \texttt{all}) \tag{3}$$

$$T(\texttt{require}(e)) ::= \texttt{require}(T_e(e)) \tag{4}$$

$$T(\texttt{while}\ e\ \{P\}) ::= \texttt{while}\ e\ \{P\} \qquad (P \text{ is fully public}) \tag{5}$$

$$T(\texttt{if}\ e\ \{P_1\}\ \texttt{else}\ \{P_2\}) ::= \texttt{if}\ T_e(e)\ \{T(P_1, T_e(e))\}\ \texttt{else}\ \{T(P_2, T_e(!e))\} \tag{6}$$

(d) Transforming statements using T.

$$T_e(c) ::= c \qquad \text{const } c \tag{7}$$

$$T_e(\text{id}) ::= \text{id} \qquad \text{var id } (\star) \tag{8}$$

$$T_e(L[e]) ::= T_L(L)[T_e(e)] \qquad \text{mapping entry} \tag{9}$$

$$T_e((e_1 + e_2)@\texttt{all}) ::= T_e(e_1) + T_e(e_2) \qquad \text{native functions} \tag{10}$$

$$T_e(\texttt{reveal}(e, \alpha)) ::= out(e, \alpha) \qquad \text{(invariant: } e@\texttt{me}) \tag{11}$$

$$T_e(e@\alpha) ::= out(e, \alpha) \qquad \text{(invariant: } e@\texttt{me}) \tag{12}$$

(e) Transforming expressions in zkay using T_e. For private function arguments id, $\star$ adds an additional correctness constraint to ϕ.

$$T_\phi(c) ::= c \qquad \text{const } c \tag{13}$$

$$T_\phi(L@\alpha) ::= in(L, \alpha) \qquad \text{(invariant: } \alpha \in \{\texttt{all}, \texttt{me}\}) \tag{14}$$

$$T_\phi(\texttt{reveal}(e, \alpha)) ::= T_\phi(e) \tag{15}$$

$$T_\phi(e_1 + e_2) ::= T_\phi(e_1) + T_\phi(e_2) \qquad \text{native functions} \tag{16}$$

(f) Transforming expressions in the proof circuit using T_ϕ.

Figure 5: Overview of zkay transformations. We write $e@\alpha$ to indicate that e has privacy type α. The symbol v_i denotes a fresh variable.

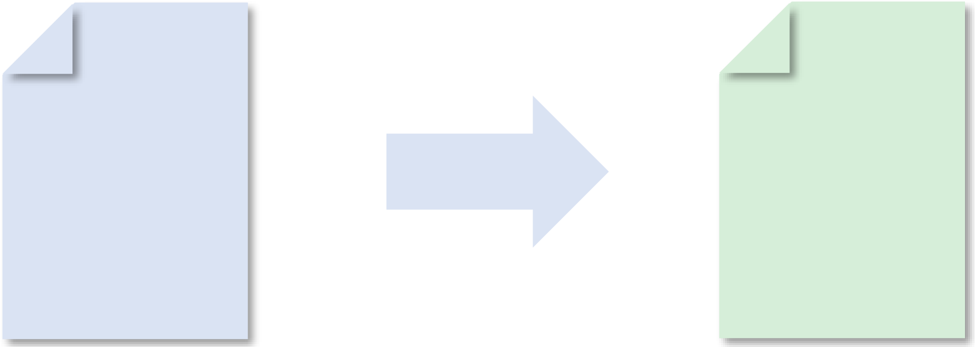
see paper for details

scattered logic

Privacy Guarantees

Data Privacy

ideal world



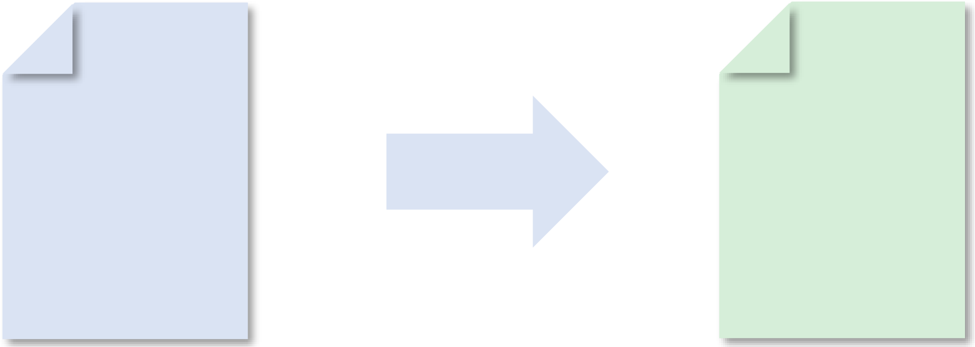
real world

input
(specification)

output
(realization)

Data Privacy

ideal world



real world

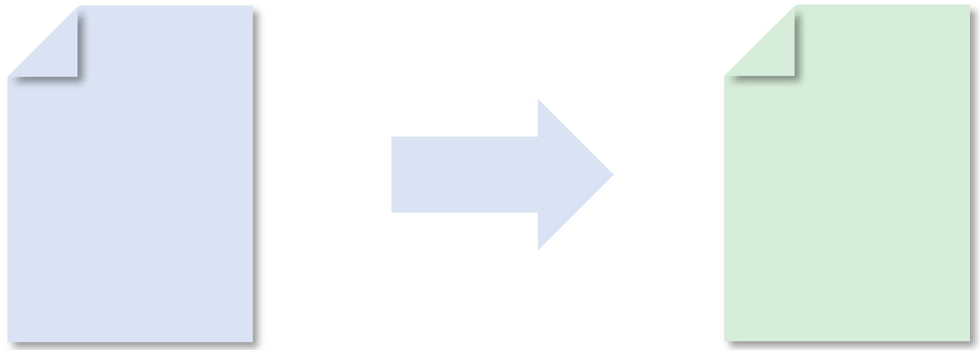
input
(specification)

output
(realization)

Everything Eve can learn here…

Data Privacy

ideal world



input
(specification)

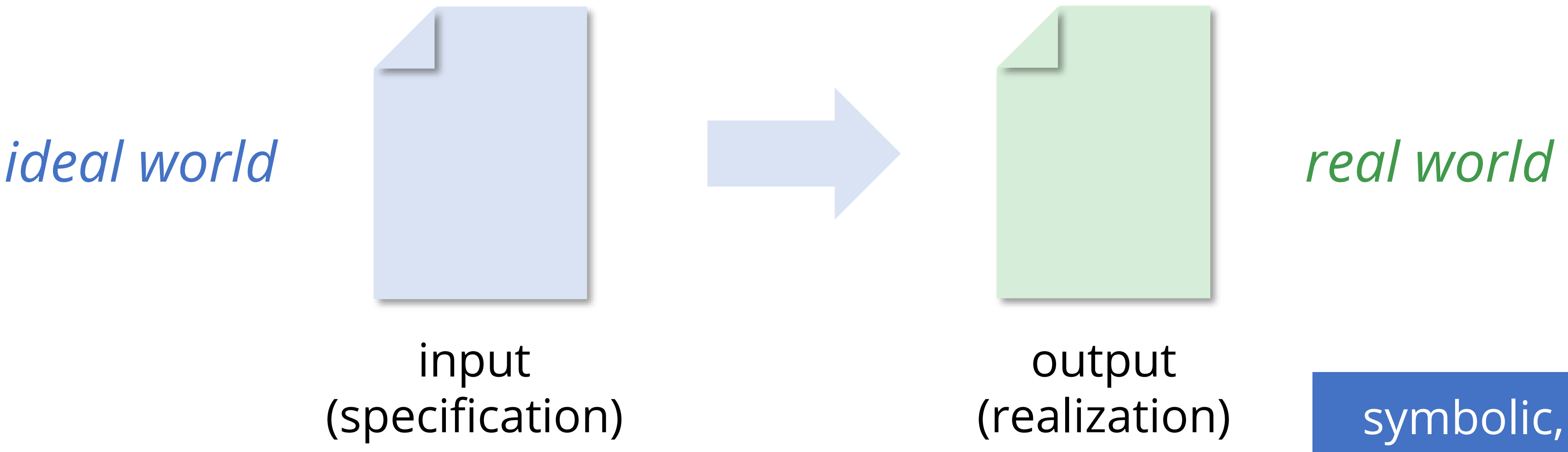
output
(realization)

real world

Everything Eve can learn here…

…she can also learn here

Data Privacy



symbolic,
Dolev-Yao style

Everything Eve can
learn here…

…she can also learn here

Data Privacy

ideal world

input
(specification)

output
(realization)

real world

symbolic,
Dolev-Yao style

Everything Eve can
learn here…

…she can also learn here

according to privacy types

Data Privacy

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

Everything Eve can learn here…

…she can also learn here

according to privacy types

Implementation

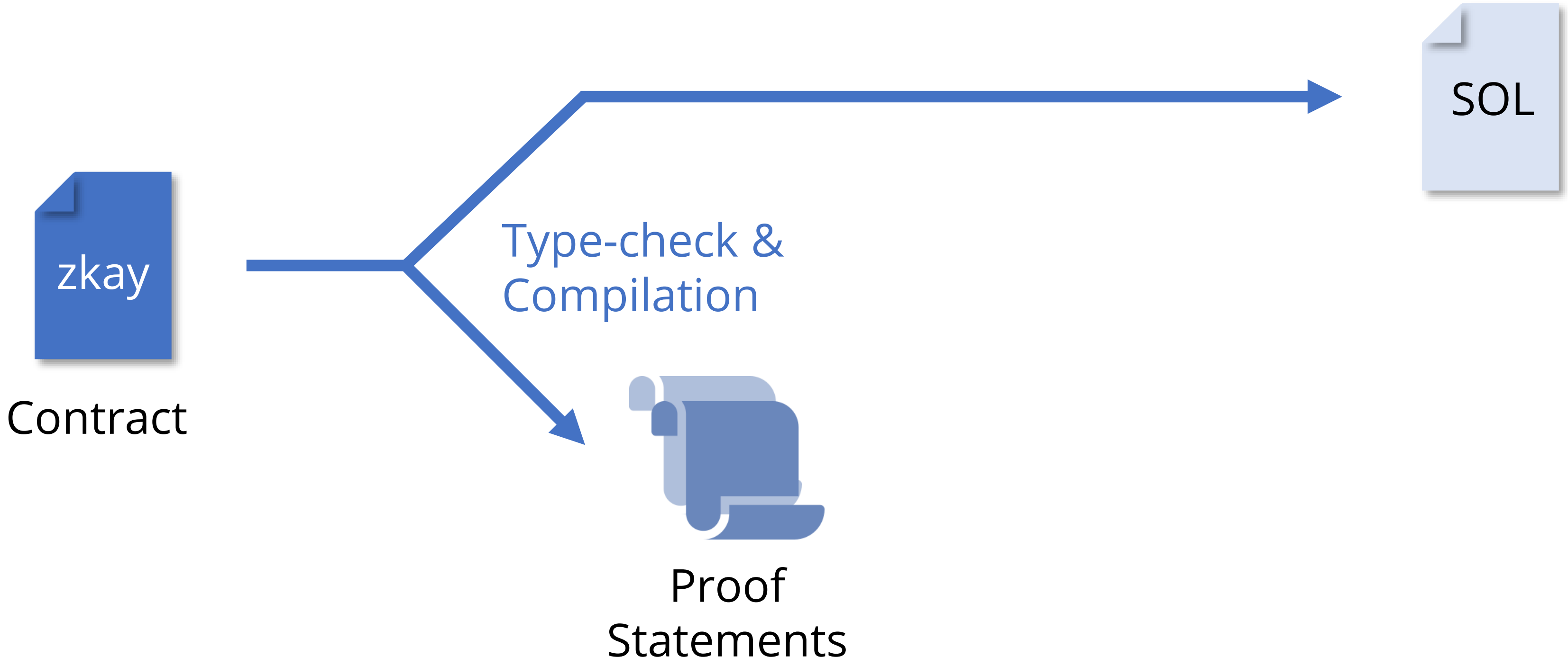
eth-sri/zkay

zkay

Contract

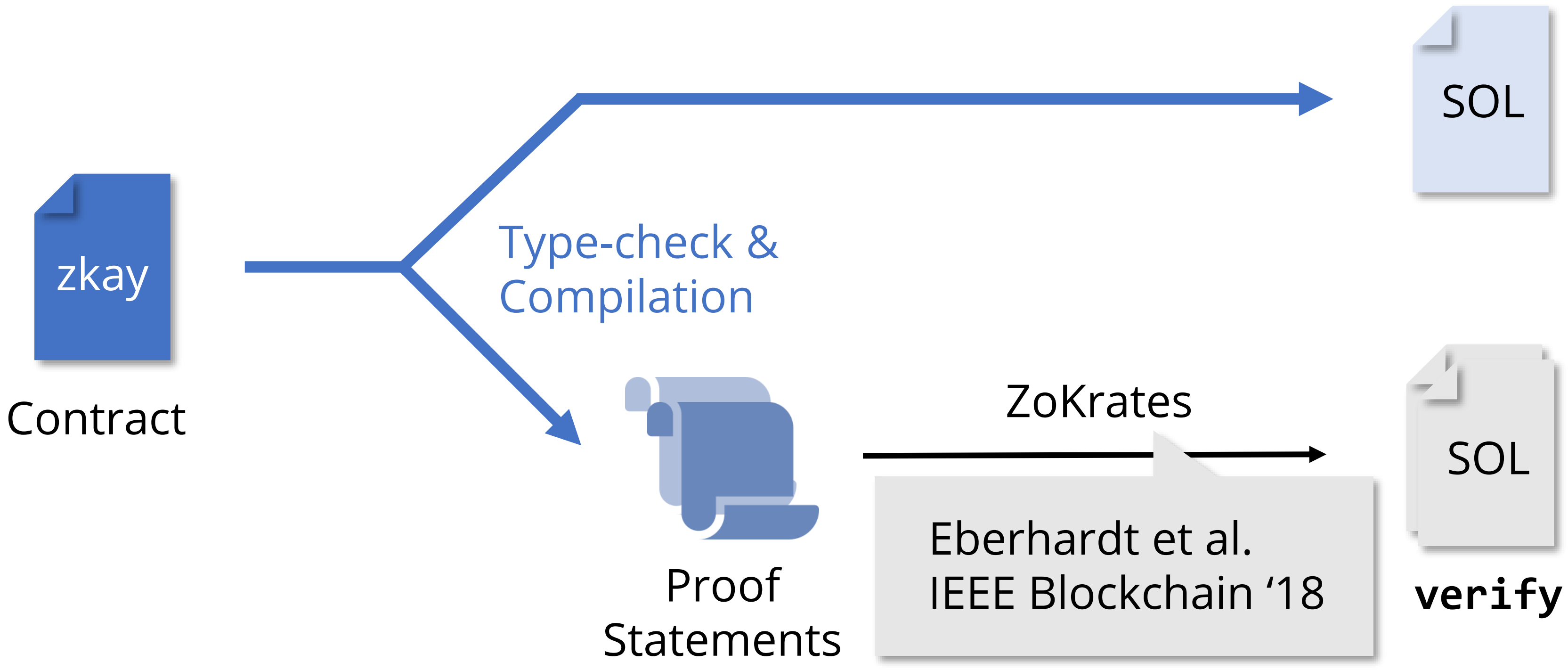
Implementation

eth-sri/zkay



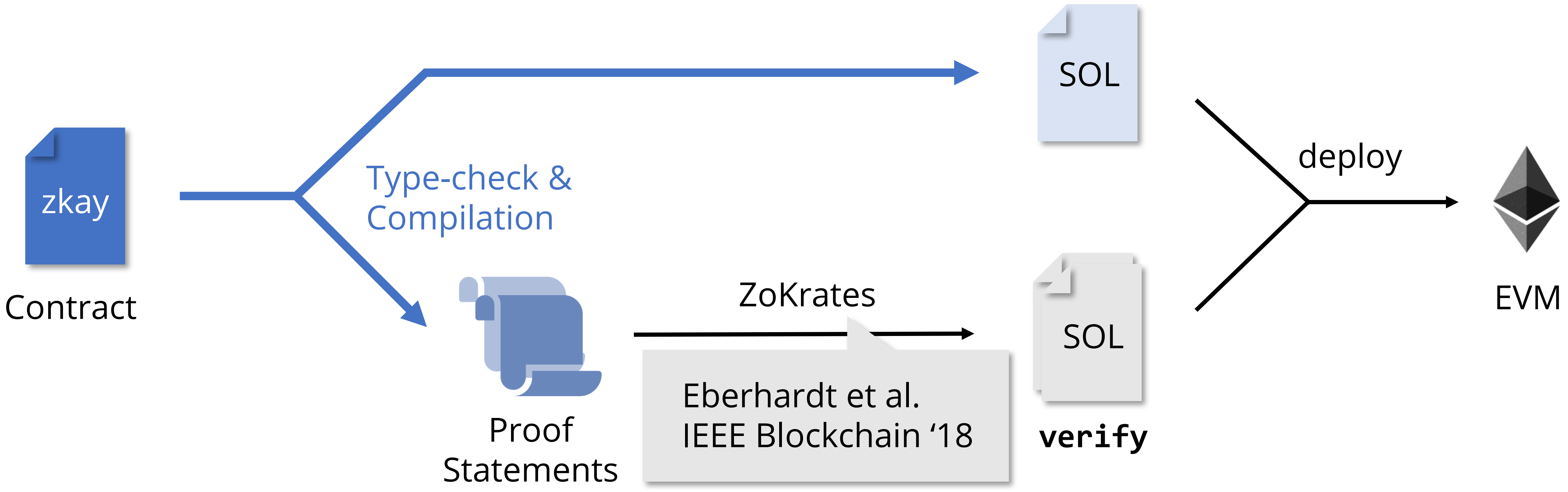
Implementation

eth-sri/zkay



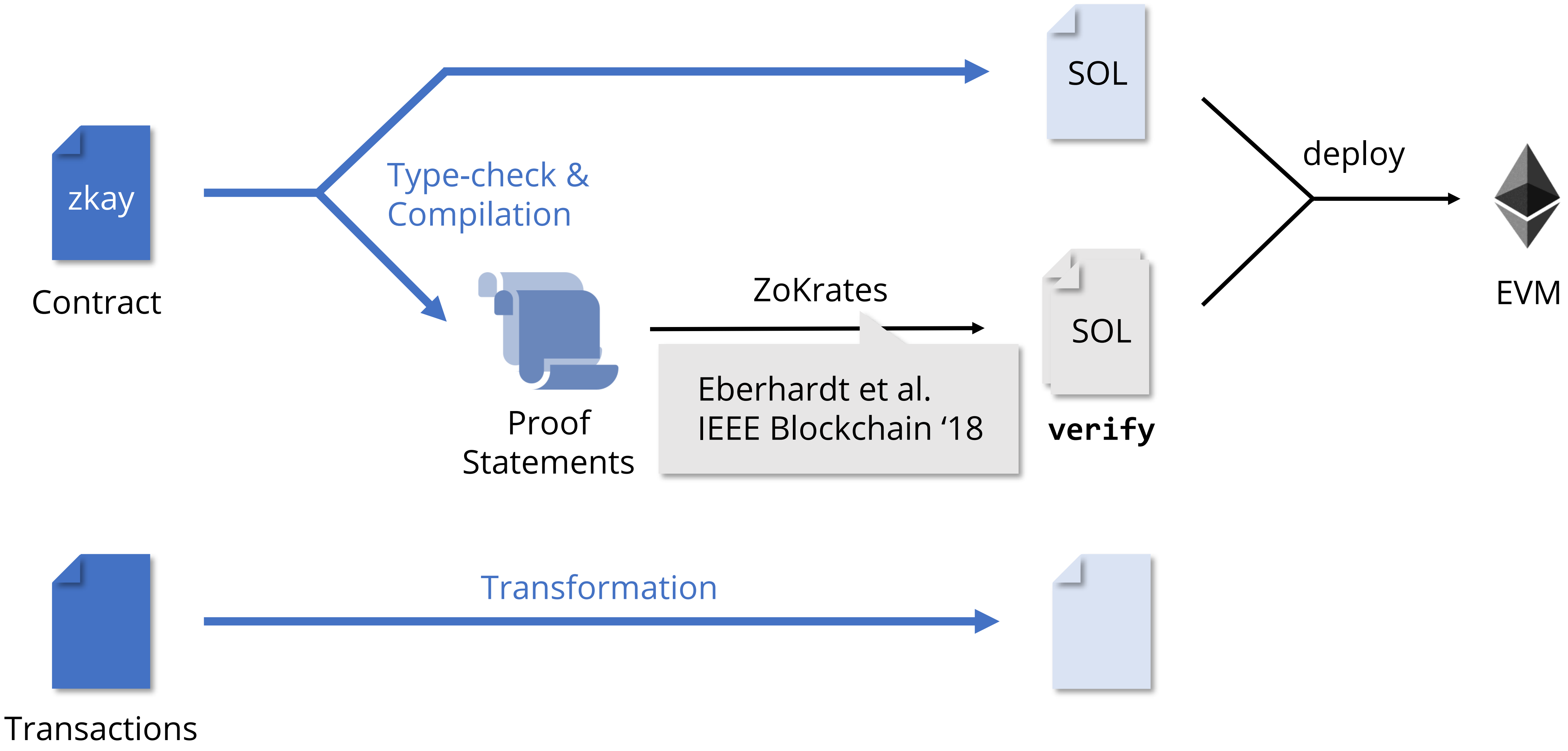
Implementation

eth-sri/zkay



Implementation

eth-sri/zkay



Evaluation

Expressiveness?

Gas costs?

Evaluation

10 Example Contracts

22 – 79 loc

Evaluation

10 Example Contracts

wide range of domains

22 – 79 loc

Insurance & Finance

Healthcare

Gambling

…

Evaluation

wide range of domains

Insurance & Finance

Healthcare

Gambling

...

10 Example Contracts

22 – 79 loc

Small example scenarios

4 – 9 transactions

Evaluation

wide range of domains

10 Example Contracts

22 – 79 loc

Insurance & Finance

Healthcare

Gambling

...

Small example scenarios

4 – 9 transactions

zkay is expressive

On-Chain Costs

Private Transactions

On-Chain Costs

Private Transactions

dominated by constant-cost*
NIZK proof verification

ca. 10^6 Gas per tx

On-Chain Costs

Private Transactions

dominated by constant-cost*
NIZK proof verification

ca. 10^6 Gas per tx

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

On-Chain Costs

Private Transactions

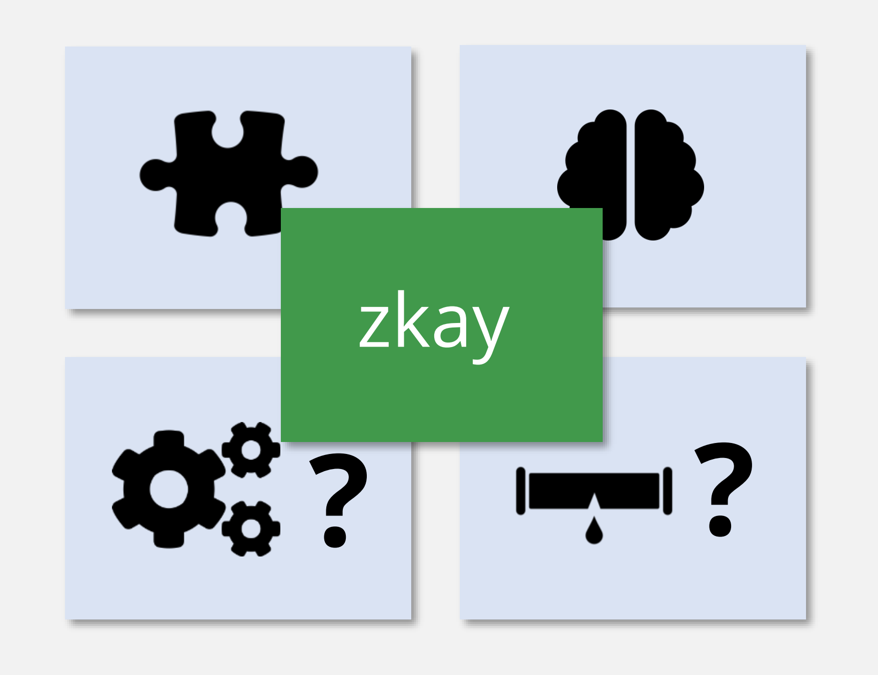
dominated by constant-cost*
NIZK proof verification

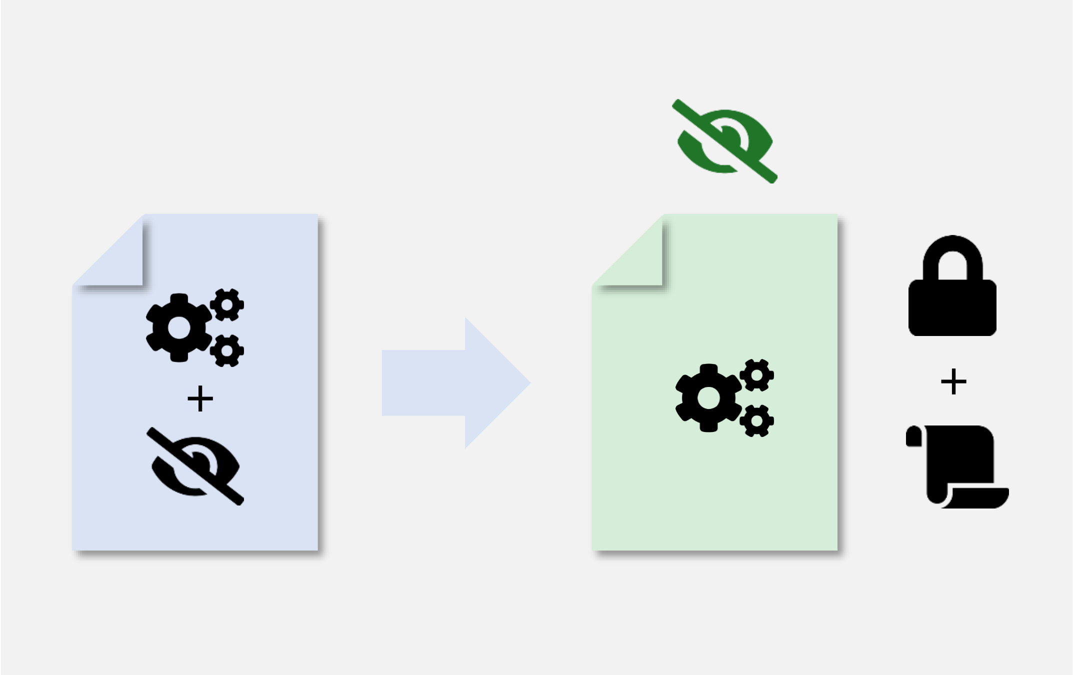
ca. 10^6 Gas per tx

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

gas costs are moderate

Summary





`datatype@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