Blockchain Privacy Preserving Techniques

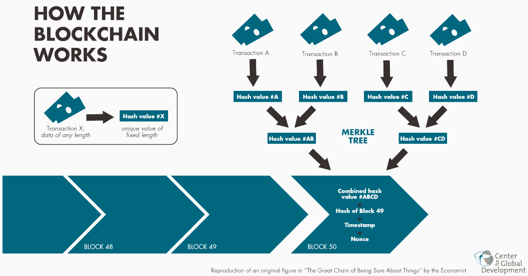
XU Cheng <chengxu@comp.hkbu.edu.hk>

October 12, 2019 @ NDBC 2019

Department of Computer Science, Hong Kong Baptist University

Blockchain Technology

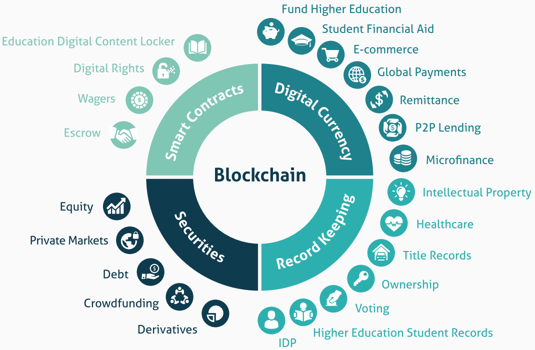
- Blockchain: Append-only data structure collectively maintained by a network of (untrusted) nodes
    - Hash chain
    - Consensus
    - Immutability
    - Decentralization



Blockchain Structure *[Credit: Wikipedia]*

Blockchain Technology

- Blockchain: Append-only data structure collectively maintained by a network of (untrusted) nodes
  - Hash chain
  - Consensus
  - Immutability
  - Decentralization
- A wide range of applications
  - Digital identities
  - Decentralized notary
  - Distributed storage
  - Smart Contracts
  - ...



Blockchain Applications *[Credit: FAHM Technology Partners]*

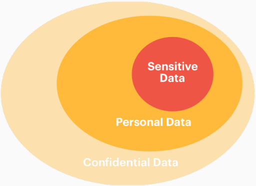
Smart Contract

- A trusted program to execute user-defined computation upon the blockchain
  - Smart Contract reads and writes blockchain data
  - Execution integrity is ensured by the consensus protocol
- Offer trusted storage and computation capabilities
- Function as a trusted virtual machine

	Traditional Computer	Blockchain VM
Storage	RAM	Blockchain
Computation	CPU	Smart Contract

Privacy Issues in Blockchain

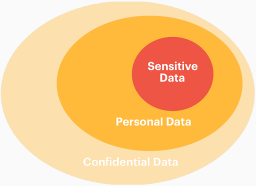
- Blockchain data is public and transparent
  - Cannot store confidential data
    - E.g., health records, bank accounts, business contracts
  - Any interaction with the smart contract is also public
  - Limit the application of blockchain technology



[Credit: Gergely Acs]

Privacy Issues in Blockchain

- Blockchain data is public and transparent
  - Cannot store confidential data
    - E.g., health records, bank accounts, business contracts
  - Any interaction with the smart contract is also public
  - Limit the application of blockchain technology



[Credit: Gergely Acs]

- Blockchain data is immutable
  - Once data is written into blockchain, it cannot be removed
  - Cannot fulfill the right to be forgotten
  - Incompatible with GDPR

[Credit: David Alayón]

Strawman Approach

- **Problem:** blockchain data is public
- **Strawman Approach**
    - Encrypt the data before writing into the blockchain

[Credit: Pixabay]

Strawman Approach

- **Problem:** blockchain data is public
- **Strawman Approach**
  - Encrypt the data before writing into the blockchain
- **Limitations**
  - Smart contract cannot process ciphertext
  - Computation can only be done locally
    - decrypt → process → encrypt
  - Encrypted computation results cannot be publicly verified
  - Access pattern still leaks confidential information

[Credit: Pixabay]

Homomorphic Encryption

- An encryption technique allows mathematical operations on plaintext to be carried out on ciphertext
  - Enable smart contract to process encrypted data directly

$$\begin{tikzcd}
\mathrm{enc}(m) \arrow[r, "\mathrm{eval}"] & \mathrm{enc}(f(m)) \\
m \arrow[u, "\mathrm{enc}"] \arrow[r, "f"] & f(m) \arrow[u, "\mathrm{enc}"']
\end{tikzcd}$$

A. Acar *et al.*, "A survey on homomorphic encryption schemes," *ACM Computing Surveys*, 2018

Homomorphic Encryption

- An encryption technique allows mathematical operations on plaintext to be carried out on ciphertext
  - Enable smart contract to process encrypted data directly

- State-of-the-art
  - Fully homomorphic encryption: Expressive but high overhead
  - Partial homomorphic encryption: Efficient but limited functions

$$
\begin{tikzcd}
\mathrm{enc}(m) \arrow[r, "\mathrm{eval}"] & \mathrm{enc}(f(m)) \\
m \arrow[u, "\mathrm{enc}"] \arrow[r, "f"] & f(m) \arrow[u, "\mathrm{enc}"']
\end{tikzcd}
$$

A. Acar *et al.*, "A survey on homomorphic encryption schemes," *ACM Computing Surveys*, 2018

Homomorphic Encryption

- An encryption technique allows mathematical operations on plaintext to be carried out on ciphertext
    - Enable smart contract to process encrypted data directly

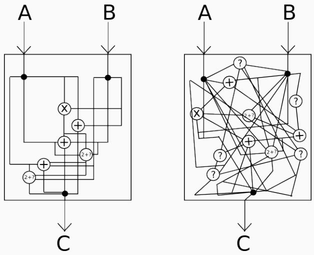
$$\begin{tikzcd} \mathrm{enc}(m) \arrow[r, "\mathrm{eval}"] & \mathrm{enc}(f(m)) \\ m \arrow[u, "\mathrm{enc}"] \arrow[r, "f"] & f(m) \arrow[u, "\mathrm{enc}"'] \end{tikzcd}$$

- State-of-the-art
    - Fully homomorphic encryption: Expressive but high overhead
    - Partial homomorphic encryption: Efficient but limited functions
- Example of partial homomorphic encryption (ElGamal)
    - $\mathrm{enc}(m) = (g^y, mh^y)$
    - $\mathrm{enc}(m_1) \cdot \mathrm{enc}(m_2) = (g^{y_1+y_2}, m_1 m_2 h^{y_1+y_2}) = \mathrm{enc}(m_1 \cdot m_2)$

A. Acar *et al.*, “A survey on homomorphic encryption schemes,” *ACM Computing Surveys*, 2018

Zero-Knowledge Proofs (ZKP)

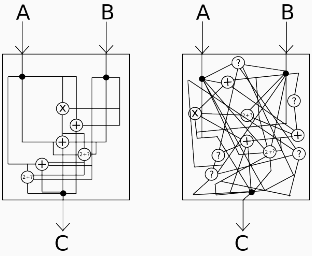
- Zero-Knowledge Proofs allow
  - Publicly verify some statement
  - Leak no information beyond the statement itself (e.g., internal states, private inputs, etc.)



[Credit: Vitalik Buterin]

A. Kosba *et al.*, “Hawk: The blockchain model of cryptography and privacy-preserving smart contracts,” in *IEEE S&P*, 2016

- Zero-Knowledge Proofs allow
  - Publicly verify some statement
  - Leak no information beyond the statement itself (e.g., internal states, private inputs, etc.)
- zk-SNARKs
  (Zero-Knowledge Succinct Non-Interactive ARguments of Knowledge)
  - Zero-Knowledge: the verifier learns nothing apart from the validity of the statement
  - Succinct: the size of the message is tiny in comparison to the length of the actual computation
  - Non-interactive: there is no or only little interaction
  - Arguments: the verifier is only protected against computationally limited provers



[Credit: Vitalik Buterin]

A. Kosba *et al.*, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *IEEE S&P*, 2016

zk-SNARKs

Program

A program can be viewed as `C(x, w) -> {0, 1}`.

- `x` is the public input.
- `w` is the secret witness input.

B. Parno *et al.*, “Pinocchio: Nearly practical verifiable computation,” in *IEEE S&P*, 2013

zk-SNARKs

Program

A program can be viewed as `C(x, w) -> {0, 1}`.

- `x` is the public input.
- `w` is the secret witness input.

Example

```
function C(x, w) { return sha256(w) == x; }
```

B. Parno *et al.*, “Pinocchio: Nearly practical verifiable computation,” in *IEEE S&P*, 2013

zk-SNARKs

Program

A program can be viewed as `C(x, w) -> {0, 1}`.

- `x` is the public input.
- `w` is the secret witness input.

Example

```
function C(x, w) { return sha256(w) == x; }
```

zk-SNARKs

zk-SNARKs consist of a tupe of PPT algorithms (KeyGen, Prove, Verify)

- $\mathrm{KeyGen}(1^\lambda, C) \to (pk, vk)$ Generate proving key pk and verification key vk for program C.
- $\mathrm{Prove}(pk, x, w) \to \pi$ Generate the proof π w.r.t. pk, x, w.
- $\mathrm{Verify}(vk, x, \pi) \to \{0, 1\}$ Output 1 iff $\exists w$ s.t. $C(x, w) = 1$.

B. Parno *et al.*, "Pinocchio: Nearly practical verifiable computation," in *IEEE S&P*, 2013

Example of Confidential Transactions

```
mapping(address => bytes32) balanceHashes;
```

- Blockchain stores balance hashes

[Credit: Christian Lundkvist]

Example of Confidential Transactions

```
mapping(address => bytes32) balanceHashes;

function senderFunction(x, w) {
  return (w.senderBalanceBefore > w.value && sha256(w.value) == x.hashValue &&
    sha256(w.senderBalanceBefore) == x.hashSenderBalanceBefore &&
    sha256(w.senderBalanceBefore - w.value) == x.hashSenderBalanceAfter);
}
```

[Credit: Christian Lundkvist]

- Blockchain stores balance hashes
- Sender proves
  - $\text{balance}_t > \text{spent}$
  - $\text{balance}_{t+1} = \text{balance}_t - \text{spent}$
  - balance_t, balance_{t+1} are well formed w.r.t. hashes

Example of Confidential Transactions

```
mapping(address => bytes32) balanceHashes;

function senderFunction(x, w) {
  return (w.senderBalanceBefore > w.value && sha256(w.value) == x.hashValue &&
    sha256(w.senderBalanceBefore) == x.hashSenderBalanceBefore &&
    sha256(w.senderBalanceBefore - w.value) == x.hashSenderBalanceAfter);
}

function receiverFunction(x, w) {
  return (sha256(w.value) == x.hashValue &&
    sha256(w.receiverBalanceBefore) == x.hashReceiverBalanceBefore &&
    sha256(w.receiverBalanceBefore + w.value) == x.hashReceiverBalanceAfter);
}
```

[Credit: Christian Lundkvist]

- Blockchain stores balance hashes
- Sender proves
  - $\text{balance}_t > \text{spent}$
  - $\text{balance}_{t+1} = \text{balance}_t - \text{spent}$
  - balance_t, balance_{t+1} are well formed w.r.t. hashes
- Recipient proves
  - $\text{balance}_{t+1} = \text{balance}_t + \text{spent}$
  - balance_t, balance_{t+1} are well formed w.r.t. hashes

Example of Confidential Transactions

```
mapping(address => bytes32) balanceHashes;

function senderFunction(x, w) {
  return (w.senderBalanceBefore > w.value && sha256(w.value) == x.hashValue &&
    sha256(w.senderBalanceBefore) == x.hashSenderBalanceBefore &&
    sha256(w.senderBalanceBefore - w.value) == x.hashSenderBalanceAfter);
}

function receiverFunction(x, w) {
  return (sha256(w.value) == x.hashValue &&
    sha256(w.receiverBalanceBefore) == x.hashReceiverBalanceBefore &&
    sha256(w.receiverBalanceBefore + w.value) == x.hashReceiverBalanceAfter);
}

function transfer(address _to, bytes32 hashValue, bytes32 hashSenderBalanceAfter,
  bytes32 hashReceiverBalanceAfter, bytes zkProofSender, bytes zkProofReceiver) {
  bytes32 hashSenderBalanceBefore = balanceHashes[msg.sender];
  bytes32 hashReceiverBalanceBefore = balanceHashes[_to];
  bool senderProofIsCorrect = zksnarkverify(confTxSenderVk,
    [hashSenderBalanceBefore, hashSenderBalanceAfter, hashValue], zkProofSender);
  bool receiverProofIsCorrect = zksnarkverify(confTxReceiverVk,
    [hashReceiverBalanceBefore, hashReceiverBalanceAfter, hashValue],
    zkProofReceiver);
  if (senderProofIsCorrect && receiverProofIsCorrect) {
    balanceHashes[msg.sender] = hashSenderBalanceAfter;
    balanceHashes[_to] = hashReceiverBalanceAfter;
  }
}
```

[Credit: Christian Lundkvist]

- Blockchain stores balance hashes
- Sender proves
  - $\text{balance}_t > \text{spent}$
  - $\text{balance}_{t+1} = \text{balance}_t - \text{spent}$
  - balance_t, balance_{t+1} are well formed w.r.t. hashes
- Recipient proves
  - $\text{balance}_{t+1} = \text{balance}_t + \text{spent}$
  - balance_t, balance_{t+1} are well formed w.r.t. hashes

Example of Confidential Transactions

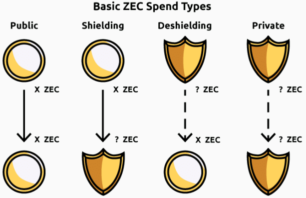
```
mapping(address => bytes32) balanceHashes;

function senderFunction(x, w) {
  return (w.senderBalanceBefore > w.value && sha256(w.value) == x.hashValue &&
    sha256(w.senderBalanceBefore) == x.hashSenderBalanceBefore &&
    sha256(w.senderBalanceBefore - w.value) == x.hashSenderBalanceAfter);
}

function receiverFunction(x, w) {
  return (sha256(w.value) == x.hashValue &&
    sha256(w.receiverBalanceBefore) == x.hashReceiverBalanceBefore &&
    sha256(w.receiverBalanceBefore + w.value) == x.hashReceiverBalanceAfter);
}

function transfer(address _to, bytes32 hashValue, bytes32 hashSenderBalanceAfter,
  bytes32 hashReceiverBalanceAfter, bytes zkProofSender, bytes zkProofReceiver) {
  bytes32 hashSenderBalanceBefore = balanceHashes[msg.sender];
  bytes32 hashReceiverBalanceBefore = balanceHashes[_to];
  bool senderProofIsCorrect = zksnarkverify(confTxSenderVk,
    [hashSenderBalanceBefore, hashSenderBalanceAfter, hashValue], zkProofSender);
  bool receiverProofIsCorrect = zksnarkverify(confTxReceiverVk,
    [hashReceiverBalanceBefore, hashReceiverBalanceAfter, hashValue],
    zkProofReceiver);
  if (senderProofIsCorrect && receiverProofIsCorrect) {
    balanceHashes[msg.sender] = hashSenderBalanceAfter;
    balanceHashes[_to] = hashReceiverBalanceAfter;
  }
}
```

[Credit: Christian Lundkvist]

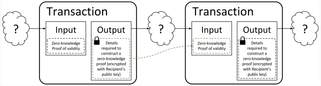
- Blockchain stores balance hashes
- Sender proves
  - $\text{balance}_t > \text{spent}$
  - $\text{balance}_{t+1} = \text{balance}_t - \text{spent}$
  - balance_t, balance_{t+1} are well formed w.r.t. hashes
- Recipient proves
  - $\text{balance}_{t+1} = \text{balance}_t + \text{spent}$
  - balance_t, balance_{t+1} are well formed w.r.t. hashes
- Drawbacks
  - Sender and recipient identities are not protected
  - Recipient need to participate transaction

Zerocash

- ZCASH uses zk-SNARKs and UTXO model to achieve unlinkable transactions
    - Transactions can be verified publicly
    - Sender, recipient and amount of a transaction remain private



[Credit: Paige Peterson]

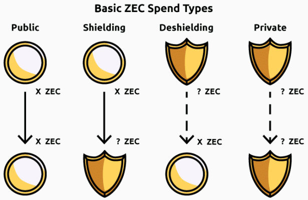


[Credit: Jack Gavigan]

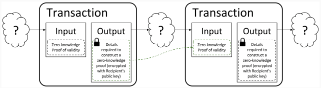
E. B. Sasson *et al.*, "Zerocash: Decentralized anonymous payments from bitcoin," in *IEEE S&P*, 2014

Zerocash

- ZCASH uses zk-SNARKs and UTXO model to achieve unlinkable transactions
  - Transactions can be verified publicly
  - Sender, recipient and amount of a transaction remain private
- Each transaction consists of inputs and outputs
  Each coin has serial number and owner address



[Credit: Paige Peterson]

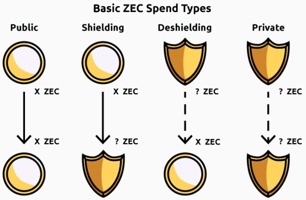


[Credit: Jack Gavigan]

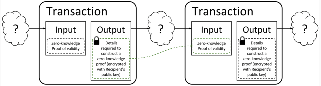
E. B. Sasson *et al.*, "Zerocash: Decentralized anonymous payments from bitcoin," in *IEEE S&P*, 2014

Zerocash

- ZCASH uses zk-SNARKs and UTXO model to achieve unlinkable transactions
  - Transactions can be verified publicly
  - Sender, recipient and amount of a transaction remain private
- Each transaction consists of inputs and outputs
  Each coin has serial number and owner address
- To spend, sender proves that in zero-knowledge
  - $\sum$ inputs $= \sum$ outputs
  - inputs $\in$ {previous outputs}
  - Sender has private keys w.r.t. inputs's owner address
  - Serial numbers are correct w.r.t. inputs's coins



[Credit: Paige Peterson]



[Credit: Jack Gavigan]

E. B. Sasson *et al.*, "Zerocash: Decentralized anonymous payments from bitcoin," in *IEEE S&P*, 2014

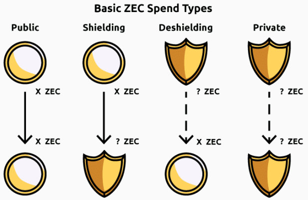
Zerocash

- ZCASH uses zk-SNARKs and UTXO model to achieve unlinkable transactions
  - Transactions can be verified publicly
  - Sender, recipient and amount of a transaction remain private

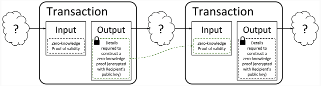
- Each transaction consists of inputs and outputs
  Each coin has serial number and owner address

- To spend, sender proves that in zero-knowledge
  - $\sum$ inputs $= \sum$ outputs
  - inputs $\in$ {previous outputs}
  - Sender has private keys w.r.t. inputs's owner address
  - Serial numbers are correct w.r.t. inputs's coins

- Miner verifies the proof and serial numbers are never spent



[Credit: Paige Peterson]

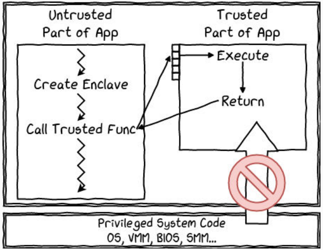


[Credit: Jack Gavigan]

E. B. Sasson *et al.*, "Zerocash: Decentralized anonymous payments from bitcoin," in *IEEE S&P*, 2014

Trusted Execution Environment

- Intel SGX (Software Guard Extension) allows to create a reverse sandbox that protects enclaves from:
  - OS or hypervisor
  - BIOS, firmware, drivers
  - Intel ME
  - Any remote attack

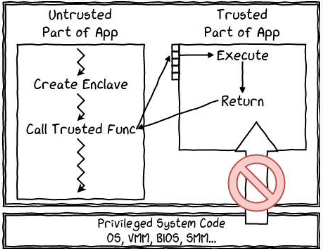


[Credit: Alexandre Adamski]

V. Costan *et al.*, *Intel SGX explained*, Cryptology ePrint Archive, Report 2016/086, 2016
R. Cheng *et al.*, "Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts," in *IEEE EuroS&P*, 2019

Trusted Execution Environment

- Intel SGX (Software Guard Extension) allows to create a reverse sandbox that protects enclaves from:
  - OS or hypervisor
  - BIOS, firmware, drivers
  - Intel ME
  - Any remote attack
- Pros
  - More efficient than zk-SNARKs
  - Support arbitrary computation tasks
  - Offer guarantees for both data integrity and confidentiality

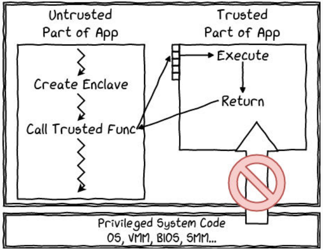


[Credit: Alexandre Adamski]

V. Costan *et al.*, *Intel SGX explained*, Cryptology ePrint Archive, Report 2016/086, 2016
R. Cheng *et al.*, "Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts," in *IEEE EuroS&P*, 2019

Trusted Execution Environment

- Intel SGX (Software Guard Extension) allows to create a reverse sandbox that protects enclaves from:
  - OS or hypervisor
  - BIOS, firmware, drivers
  - Intel ME
  - Any remote attack
- Pros
  - More efficient than zk-SNARKs
  - Support arbitrary computation tasks
  - Offer guarantees for both data integrity and confidentiality
- Cons
  - Hardware instead of cryptographic based security guarantee
  - You need to trust Intel (a centralized party)
  - Recent attacks through Spectre and Meltdown



[Credit: Alexandre Adamski]

V. Costan *et al.*, *Intel SGX explained*, Cryptology ePrint Archive, Report 2016/086, 2016
R. Cheng *et al.*, "Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts," in *IEEE EuroS&P*, 2019

Oblivious Data Access

- Side-Channel Attack
  - Data access pattern can leak critical information

K. Nayak *et al.*, "GraphSC: Parallel secure computation made easy," in *IEEE S&P*, 2015
E. Cecchetti *et al.*, "Solidus: Confidential distributed ledger transactions via PVORM," in *ACM CCS*, 2017

Oblivious Data Access

- Side-Channel Attack
    - Data access pattern can leak critical information
- Oblivious Algorithms
    - Process data in oblivious manner
    - Tailor to the specific task, relatively efficient
    - Example: oblivious sort, oblivious join, oblivious graph query processing, etc.

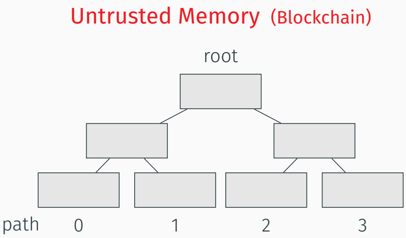
K. Nayak *et al.*, "GraphSC: Parallel secure computation made easy," in *IEEE S&P*, 2015
E. Cecchetti *et al.*, "Solidus: Confidential distributed ledger transactions via PVORM," in *ACM CCS*, 2017

Oblivious Data Access

- Side-Channel Attack
  - Data access pattern can leak critical information
- Oblivious Algorithms
  - Process data in oblivious manner
  - Tailor to the specific task, relatively efficient
  - Example: oblivious sort, oblivious join, oblivious graph query processing, etc.
- Oblivious RAM (ORAM)
  - General memory access model: $\texttt{Read}(k), \texttt{Write}(k, v)$
  - Allow access the data in arbitrary orders
  - Leak no information from the access pattern

K. Nayak *et al.*, "GraphSC: Parallel secure computation made easy," in *IEEE S&P*, 2015
E. Cecchetti *et al.*, "Solidus: Confidential distributed ledger transactions via PVORM," in *ACM CCS*, 2017

Path ORAM



- Data Structures
  - Untrusted memory is structured as a binary tree

E. Stefanov *et al.*, "Path ORAM: An extremely simple oblivious RAM protocol," in *ACM CCS*, 2013

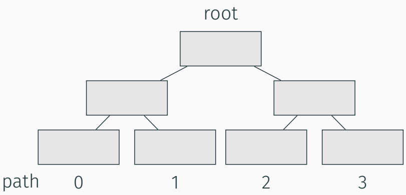
Trusted Memory (Client)

Stash

Position Map

Block	B_0	B_1	B_2	B_3	B_4
Path					

Untrusted Memory (Blockchain)



- Data Structures
  - Untrusted memory is structured as a binary tree
  - Trusted memory consists of
    - Position Map: map block to a random path
    - Stash: temporary storage

E. Stefanov *et al.*, "Path ORAM: An extremely simple oblivious RAM protocol," in *ACM CCS*, 2013

Path ORAM

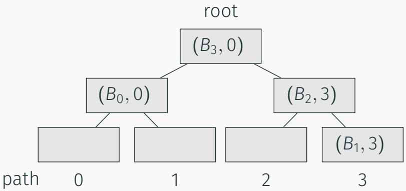
Trusted Memory (Client)

Stash

Position Map

Block	B_0	B_1	B_2	B_3	B_4
Path	0	3	3	0	1

Untrusted Memory (Blockchain)



- Data Structures
  - Untrusted memory is structured as a binary tree
  - Trusted memory consists of
    - Position Map: map block to a random path
    - Stash: temporary storage
  - A block is stored in the untrusted memory or stash

E. Stefanov *et al.*, “Path ORAM: An extremely simple oblivious RAM protocol,” in *ACM CCS*, 2013

Path ORAM

Trusted Memory (Client)

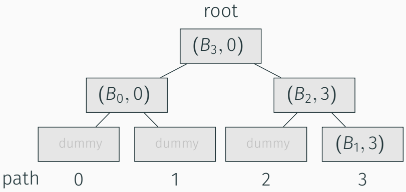
Stash

$(B_4, 1)$			

Position Map

Block	B_0	B_1	B_2	B_3	B_4
Path	0	3	3	0	1

Untrusted Memory (Blockchain)



- Data Structures
  - Untrusted memory is structured as a binary tree
  - Trusted memory consists of
    - Position Map: map block to a random path
    - Stash: temporary storage
  - A block is stored in the untrusted memory or stash
  - Unused untrusted memory is filled with dummy block

E. Stefanov *et al.*, "Path ORAM: An extremely simple oblivious RAM protocol," in *ACM CCS*, 2013

Path ORAM

Trusted Memory (Client)

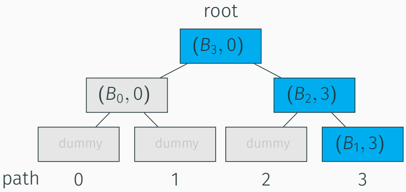
Stash

$(B_4, 1)$			

Position Map

Block	B_0	B_1	B_2	B_3	B_4
Path	0	3	3	0	1

Untrusted Memory (Blockchain)



- **Data Structures**
  - Untrusted memory is structured as a binary tree
  - Trusted memory consists of
    - Position Map: map block to a random path
    - Stash: temporary storage
  - A block is stored in the untrusted memory or stash
  - Unused untrusted memory is filled with dummy block
- **Access Block B_1**
  - Lookup position map to locate the block B_1

E. Stefanov *et al.*, "Path ORAM: An extremely simple oblivious RAM protocol," in *ACM CCS*, 2013

Path ORAM

Trusted Memory (Client)

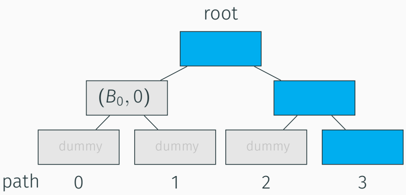
Stash

$(B_4, 1)$	$(B_1, 3)$	$(B_2, 3)$	$(B_3, 0)$

Position Map

Block	B_0	B_1	B_2	B_3	B_4
Path	0	3	3	0	1

Untrusted Memory (Blockchain)



- **Data Structures**
  - Untrusted memory is structured as a binary tree
  - Trusted memory consists of
    - Position Map: map block to a random path
    - Stash: temporary storage
  - A block is stored in the untrusted memory or stash
  - Unused untrusted memory is filled with dummy block

- **Access Block B_1**
  - Lookup position map to locate the block B_1
  - Read all blocks on path 3 to the stash

E. Stefanov *et al.*, "Path ORAM: An extremely simple oblivious RAM protocol," in *ACM CCS*, 2013

Path ORAM

Trusted Memory (Client)

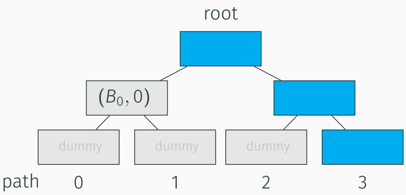
Stash

$(B_4, 1)$	$(B_1, 3)$	$(B_2, 3)$	$(B_3, 0)$

Position Map

Block	B_0	B_1	B_2	B_3	B_4
Path	0	3	3	0	1

Untrusted Memory (Blockchain)



- **Data Structures**
  - Untrusted memory is structured as a binary tree
  - Trusted memory consists of
    - Position Map: map block to a random path
    - Stash: temporary storage
  - A block is stored in the untrusted memory or stash
  - Unused untrusted memory is filled with dummy block

- **Access Block B_1**
  - Lookup position map to locate the block B_1
  - Read all blocks on path 3 to the stash
  - Apply operation

E. Stefanov *et al.*, “Path ORAM: An extremely simple oblivious RAM protocol,” in *ACM CCS*, 2013

Path ORAM

Trusted Memory (Client)

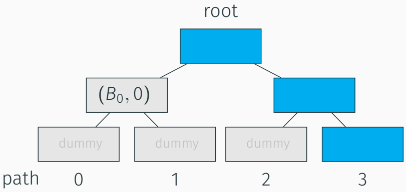
Stash

$(B_4, 1)$	$(B_1, 1)$	$(B_2, 3)$	$(B_3, 0)$

Position Map

Block	B_0	B_1	B_2	B_3	B_4
Path	0	1	3	0	1

Untrusted Memory (Blockchain)



- **Data Structures**
  - Untrusted memory is structured as a binary tree
  - Trusted memory consists of
    - Position Map: map block to a random path
    - Stash: temporary storage
  - A block is stored in the untrusted memory or stash
  - Unused untrusted memory is filled with dummy block

- **Access Block B_1**
  - Lookup position map to locate the block B_1
  - Read all blocks on path 3 to the stash
  - Apply operation
  - Remap B_1 to a new random path

E. Stefanov *et al.*, "Path ORAM: An extremely simple oblivious RAM protocol," in *ACM CCS*, 2013

Path ORAM

Trusted Memory (Client)

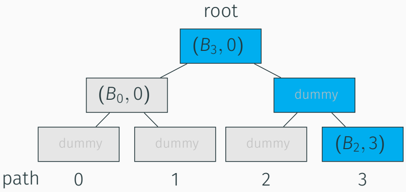
Stash

$(B_4, 1)$	$(B_1, 1)$		

Position Map

Block	B_0	B_1	B_2	B_3	B_4
Path	0	1	3	0	1

Untrusted Memory (Blockchain)



- Data Structures
  - Untrusted memory is structured as a binary tree
  - Trusted memory consists of
    - Position Map: map block to a random path
    - Stash: temporary storage
  - A block is stored in the untrusted memory or stash
  - Unused untrusted memory is filled with dummy block
- Access Block B_1
  - Lookup position map to locate the block B_1
  - Read all blocks on path 3 to the stash
  - Apply operation
  - Remap B_1 to a new random path
  - Write as many blocks as possible back to path 3

E. Stefanov *et al.*, "Path ORAM: An extremely simple oblivious RAM protocol," in *ACM CCS*, 2013

Redactable Blockchain

- Fulfill the right to be forgotten

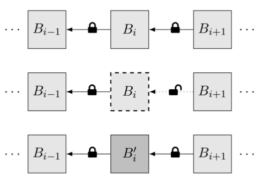
G. Ateniese *et al.*, “Redactable blockchain–or–rewriting history in bitcoin and friends,” in *IEEE EuroS&P*, 2017
D. Derler *et al.*, “Fine-grained and controlled rewriting in blockchains: Chameleon-hashing gone attribute-based,” in *NDSS*, 2019

Redactable Blockchain

- Fulfill the right to be forgotten

- Chameleon Hash Function allows authorized party to generate hash collisions

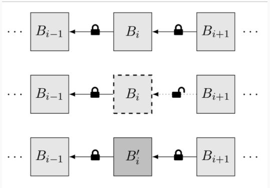
  - $\mathsf{CHGen}(1^\lambda) \rightarrow (csk, cpk)$: generate key pair (csk, cpk)
  - $\mathsf{Ch}(m; r) \rightarrow hash$: on input message m and some randomness r, output a hash value $hash$
  - $\mathsf{Col}(csk, m, r, m') \rightarrow r'$: on input secret key csk, old message m, old randomness r and a new message m', output r' such that $\mathsf{Ch}(m; r) = \mathsf{Ch}(m'; r')$



G. Ateniese *et al.*, "Redactable blockchain–or–rewriting history in bitcoin and friends," in *IEEE EuroS&P*, 2017
D. Derler *et al.*, "Fine-grained and controlled rewriting in blockchains: Chameleon-hashing gone attribute-based," in *NDSS*, 2019

Redactable Blockchain

- Fulfill the right to be forgotten
- Chameleon Hash Function allows authorized party to generate hash collisions
  - $\mathsf{CHGen}(1^\lambda) \to (csk, cpk)$: generate key pair (csk, cpk)
  - $\mathsf{Ch}(m; r) \to hash$: on input message m and some randomness r, output a hash value $hash$
  - $\mathsf{Col}(csk, m, r, m') \to r'$: on input secret key csk, old message m, old randomness r and a new message m', output r' such that $\mathsf{Ch}(m; r) = \mathsf{Ch}(m'; r')$



- The secret key is shared among miners using secret shares. When there are enough consensus to overwrite a block, multi-party computation is used to compute the updated block.

G. Ateniese *et al.*, "Redactable blockchain–or–rewriting history in bitcoin and friends," in *IEEE EuroS&P*, 2017
D. Derler *et al.*, "Fine-grained and controlled rewriting in blockchains: Chameleon-hashing gone attribute-based," in *NDSS*, 2019

Redactable Blockchain

- Issues
  - Cannot distinguish between normal block and redacted block
  - Requires heavily cryptographic operation
  - System involves trapdoor keys
  - Original miners control the redaction process

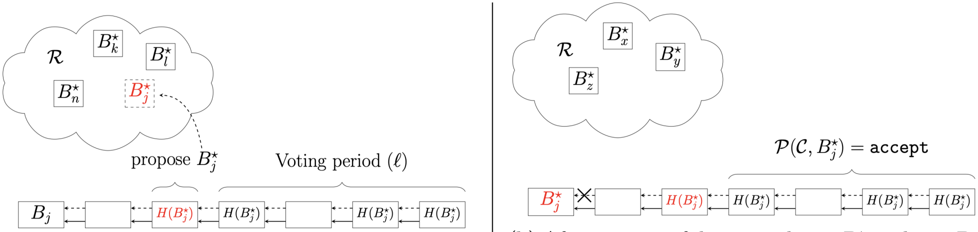
[Credit: Pixabay]

Redactable Blockchain

- Issues
  - Cannot distinguish between normal block and redacted block
  - Requires heavily cryptographic operation
  - System involves trapdoor keys
  - Original miners control the redaction process
- Desired Features
  - Make redaction transparent and accountable
  - Avoid using multi-party computation
  - Avoid introducing secret keys
  - Current miners control the redaction process

[Credit: Pixabay]

- Redaction procedure consists of: proposal → vote → accept



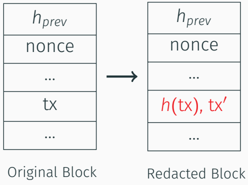
(a) Proposing a redaction $B_j^\star$ for the block B_j

(b) After a successful voting phase, $B_j^\star$ replaces B_j in the chain

D. Deuber *et al.*, "Redactable blockchain in the permissionless setting," in *IEEE S&P*, 2019

Redactable Blockchain

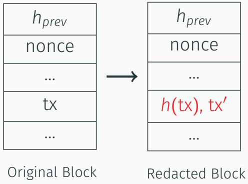
- If accepting redaction:
  - Replace redacted transaction to its hash
  - Add updated transaction



D. Deuber *et al.*, "Redactable blockchain in the permissionless setting," in *IEEE S&P*, 2019

Redactable Blockchain

- If accepting redaction:
  - Replace redacted transaction to its hash
  - Add updated transaction
- To validate the redacted block:
  - $h_{\text{old blk}} = H(h_{\text{prev}}|nonce|\text{original Merkle root})$
  - $h_{\text{new blk}} = H(h_{\text{prev}}|nonce|\text{updated Merkle root})$
  - Check consensus protocol (e.g. PoW, PoS) with respect to $h_{\text{old blk}}$
  - Check redaction block ($h_{\text{new blk}}$) was approved by policy $\mathcal{P}$
  - Check validity of data in block



D. Deuber *et al.*, "Redactable blockchain in the permissionless setting," in *IEEE S&P*, 2019

Open Problems

- Search on encrypted blockchain data
- Data sharing with fine-grained access control
- Data integrity meets confidentiality
- Security and privacy for off-chain storage
- ...

[Credit: Pixabay]

Thanks

Questions?

References

[AAUC18] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, "A survey on homomorphic encryption schemes," *ACM Computing Surveys*, 2018.

[AMVA17] G. Ateniese, B. Magri, D. Venturi, and E. Andrade, "Redactable blockchain–or–rewriting history in bitcoin and friends," in *IEEE EuroS&P*, 2017.

[CD16] V. Costan and S. Devadas, *Intel SGX explained*, Cryptology ePrint Archive, Report 2016/086, 2016.

[CZJ+17] E. Cecchetti, F. Zhang, Y. Ji, A. Kosba, A. Juels, and E. Shi, "Solidus: Confidential distributed ledger transactions via PVORM," in *ACM CCS*, 2017.

[CZK+19] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song, "Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts," in *IEEE EuroS&P*, 2019.

[DMT19] D. Deuber, B. Magri, and S. A. K. Thyagarajan, "Redactable blockchain in the permissionless setting," in *IEEE S&P*, 2019.

[DSSS19] D. Derler, K. Samelin, D. Slamanig, and C. Striecks, "Fine-grained and controlled rewriting in blockchains: Chameleon-hashing gone attribute-based," in *NDSS*, 2019.

[KMS+16] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *IEEE S&P*, 2016.

References

[NWI+15] K. Nayak, X. S. Wang, S. Ioannidis, U. Weinsberg, N. Taft, and E. Shi, “GraphSC: Parallel secure computation made easy,” in *IEEE S&P*, 2015.

[PHGR13] B. Parno, J. Howell, C. Gentry, and M. Raykova, “Pinocchio: Nearly practical verifiable computation,” in *IEEE S&P*, 2013.

[SCG+14] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, “Zerocash: Decentralized anonymous payments from bitcoin,” in *IEEE S&P*, 2014.

[SVS+13] E. Stefanov, M. Van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas, “Path ORAM: An extremely simple oblivious RAM protocol,” in *ACM CCS*, 2013.