Cryptography in Blockchain Part II: Zero Knowledge Proof and ZK-SNARK

Zhiguo Wan
Shandong University
Outline

Crypto Basics → Monero and RingCT → Zero Knowledge Proof → ZeroCash and ZK-SNARK
Mathematical Proofs

• When referring to a proof in logic we usually mean:
  • 1. Based on axioms.
  • 2. A sequence of statements.
  • 3. Each statement is derived via the derivation rules.
  • 4. The proof is fixed, i.e, in any time, anyone can read it, and get convinced.

• Or, we “prove” a statement by convincing someone.
  • For example, in court the prosecutor tries to convince the judge that the defendant is guilty.
  • The prosecutor challenges the defendant.
  • In case he fails to answer in a consistent manner, we say that the prosecutor proved his point.

Source from Eli Biham
Zero-Knowledge Proof

• About proving somebody knows some secret without revealing it

• **Interactive:** between the Prover & Verifier (P and V or Peggy & Victor)

• Does not give away any information about the secret (zero-knowledge)

• Does not allow the verifier to impersonate the prover
A very simple scenario

- Secret: the key to open the door (C)
- Uses *cut-and-choose* protocol
- Enough to prove to Victor
- Not enough to convince a third party (Carol) of proof’s validity
- *(Implicit) Commitment:* enter cannel from left or right; cannot change!!
Zero Knowledge Proof

Knowledge Types

• Statements about “facts”
  • “a specific graph has a three coloring”
  • “some number N is in the set of composite numbers“.
  • Each of these is a statement about some intrinsic property of the universe.
  • This is What

• Statements about personal knowledge.
  • “I know a three coloring for this graph”
  • “I know the factorization of N”
  • These go beyond merely proving that a fact is true, and actually rely on what the Prover knows.
  • This is Why
Example: 3-coloring
Graph-Isomorphism

• If two graphs are identical except for the names of the points, they are called isomorphic.

• Peggy knows the Isomorphism between two big Graphs $G_1$ and $G_2$

• (Commitment) Peggy generates graphs $H_1$, $H_2$, …, $H_n$, which are isomorphic to $G_1$ and $G_2$

• Victor asks Peggy to show isomorphism of $H_i$ with either $G_1$ or $G_2$, but not both

• Why Zero-knowledge?
  • NP problem!
Hamiltonian Cycles

• A circular, continuous path along the lines of a graph that passes through each point exactly once

• (1) Peggy randomly permutes G to make a new graph, H. She then encrypts H to get H′.

• (2) Peggy gives Victor a copy of H′.

• (3) Victor asks Peggy either to:
  • (a) prove to him that H′ is an encryption of an isomorphic copy of G, or
  • (b) show him a Hamiltonian cycle for H.

• (4) Peggy complies. She either:
  • (a) proves that H′ is an encryption of an isomorphic copy of, or
  • (b) shows a Hamiltonian cycle for H.

• (5) Peggy and Victor repeat steps (1) through (4) n times.
Zero Knowledge Proof Properties

• **Completeness:** If the Prover is honest, then she will eventually convince the Verifier.

• **Soundness:** The Prover can only convince the Verifier if the statement is true.

• **Zero-knowledge(ness):** The Verifier learns no information beyond the fact that the statement is true.
Zero Knowledge Proof
Fiat - Shamir protocol

• \( n = pq, \ v = s^2 \ mod \ (n-1) \)
• Prove knowledge of \( s \) given \( v \).
• \( A \) chooses \( r \) and sends \( x = r^2 \ mod \ n \) to \( B \).
• \( B \) sends \( e = 0 \) or \( 1 \) to \( A \)
• \( A \) computes \( y = r*s^e \ mod \ n \) and sends to \( B \).
• \( B \) checks that \( y^2 = x*v^e \ mod \ p \).
• **Modular Quadratic Root**
Zero Knowledge Proof of secret exponent

• Given a value $y$, a large prime $p$ and a generator $g$, she wants to prove that she knows a value $x$ such that $g^x \mod p = y$, without revealing $x$.

• in each round, Peggy generates a random number $r$, computes $C = g^r \mod p$ and discloses this to Victor.

• Victor requests $r$ or $(x+r) \mod p-1$
  • Verify $C = g^r$ if $r$ is received
  • Verify $C \cdot y = g^{x+r} \mod p$
Parallel Zero-Knowledge Proofs

• $n$ interactions to parallel execution?

• (1) Peggy uses her information and $n$ random numbers to transform the hard problem into $n$ different isomorphic problems. She then uses her information and the random numbers to solve the $n$ new hard problems.

• (2) Peggy commits to the solution of the $n$ new hard problems.

• (3) Peggy reveals to Victor the $n$ new hard problems.

• (4) For each of the $n$ new hard problems, Victor asks Peggy either to:
  • (a) prove to him that the old and new problems are isomorphic, or
  • (b) open the solution she committed to in step (2) and prove that it is a solution to the new problem.

• (5) Peggy complies for each of the $n$ new hard problems.

• Can we replace (4) with non-interactive operations?
Non-Interactive ZK Proof

• Uncertainty about “which question would be asked” is the backbone of ZKPS

• Interaction from Victor provides the random choice, or the element of uncertainty

• Use one-way (unpredictable) function as Victor surrogate

• Number of rounds must be large enough

  • (3) Peggy uses all of these commitments together as a single input to a one-way hash function. She then saves the first $n$ bits of the output of this one-way hash function.

  • (4) Peggy takes the $n$ bits generated in step (3). For each $i$th new hard problem in turn, she takes the $i$th bit of those $n$ bits and:
    • (a) if it is a 0, she proves that the old and new problems are isomorphic, or
    • (b) if it is a 1, she opens the solution she committed to in step (2) and proves that it is a solution to the new problem.
Zerocoin: A more relevant example (IEEE S&P 2013)

• Building blocks:
  • One-way Accumulator: Proving one value in a set
  • Non-interactive Zero Knowledge Proof

• Overview of Zerocoin
  • Mint: convert a Bitcoin into Zerocoin by \( c = g^S h^r \mod p \), where \( S, r \) is kept secretly.
  • Spend: from Zerocoin to Bitcoin. Providing a zero knowledge proof \((\pi, S)\) of \( r \) for Zerocoin \( c \).
  • Note: double spending is prevented by \( S \), whose reuse can be immediately detected.

• Problems:
  • A decentralized mix
  • Amount exposed
  • ZKP: 45kB and 450ms for verification
Outline

- Crypto Basics
- Monero and RingCT
- Zero Knowledge Proof
- ZeroCash and ZK-SNARK
zk-SNARK

- Zero knowledge
- Succinct
- Non-interactive

Argument
- Computationally sound proof v.s. perfect soundness
- Soundness: it is impossible to "prove" a false assertion

Of Knowledge
zk-SNARK: A Simple Example

• Arithmetic equation: \( x^3 + x + 5 = y \)
• Proof: know an \( x \) satisfying the equation
• Step 1: from equation to arithmetic circuit

\[
\begin{align*}
    y_3 + & \quad 5 \\
    y_2 + & \quad y_1 \\
    y_1 \times & \quad x \\
    x \times & \quad x \\
x \times & \quad x
\end{align*}
\]

• \( x^2 = y_1 \)
• \( y_1 \times x = y_2 \)
• \( y_2 + x = y_3 \)
• \( y_3 + 5 = y \)
zk-SNARK: A Simple Example (2)

• From Circuit to R1CS
  • \( x^2 = y_1 \)
  • \( y_1 \cdot x = y_2 \)
  • \( y_2 + x = y_3 \)
  • \( y_3 + 5 = y \)
  • \( S = [1, x, y, y_1, y_2, y_3] \)
  • \( x = [0, 1, 0, 0, 0, 0] \cdot S \)
  • \( y = [0, 0, 1, 0, 0, 0] \cdot S \)
  • \( y_1 = [0, 0, 0, 1, 0, 0] \cdot S \)
  • \( y_2 = [0, 0, 0, 0, 1, 0] \cdot S \)
  • \( y_3 = [0, 0, 0, 0, 1] \cdot S \)

• R1CS: rank-1 constraint system \((a, b, c)\)
  • A solution \( s \) satisfies R1CS if:
    \[ s \cdot a \cdot s \cdot b - s \cdot c = 0 \]
  • \( a = [0, 1, 0, 0, 0, 0] \)
  • \( b = [0, 1, 0, 0, 0, 0] \)
  • \( c = [0, 0, 0, 1, 0, 0] \)
  • \( a = [0, 0, 0, 1, 0, 0] \)
  • \( b = [0, 1, 0, 0, 0, 0] \)
  • \( c = [0, 0, 0, 0, 1] \)
  • \( a = [5, 0, 0, 0, 0, 1] \)
  • \( b = [1, 0, 0, 0, 0, 0] \)
  • \( c = [0, 0, 1, 0, 0, 0] \)
zk-SNARK: A Simple Example (3)

• The solution S should satisfy all the four constraints

• Can satisfy them in one shot? **Quadratic Arithmetic Program (QAP)**

\[
\begin{align*}
  a &= [0, 1, 0, 0, 0, 0] \\
  b &= [0, 1, 0, 0, 0, 0] \\
  c &= [0, 0, 0, 0, 1, 0] \\
  a &= [0, 0, 0, 1, 0, 0] \\
  b &= [0, 1, 0, 0, 0, 0] \\
  c &= [0, 0, 0, 0, 1, 0] \\
  a &= [0, 1, 0, 0, 1, 0] \\
  b &= [1, 0, 0, 0, 0, 0] \\
  c &= [0, 0, 0, 0, 1, 0] \\
  a &= [5, 0, 0, 0, 0, 1] \\
  b &= [1, 0, 0, 0, 0, 0] \\
  c &= [0, 0, 1, 0, 0, 0]
\end{align*}
\]

\[
\begin{align*}
  A: [0, 1, 0, 0, 0, 0] \\
  B: [0, 0, 0, 1, 0, 0] \\
  C: [0, 1, 0, 0, 1, 0] \\
  A(1) &= [0, 1, 0, 0, 0, 0] \\
  A(2) &= [0, 0, 0, 1, 0, 0] \\
  A(3) &= [0, 1, 0, 0, 1, 0] \\
  A(4) &= [5, 0, 0, 0, 0, 1] \\
  B(1) &= [0, 1, 0, 0, 0, 0] \\
  B(2) &= [0, 1, 0, 0, 0, 0] \\
  B(3) &= [1, 0, 0, 0, 0, 0] \\
  B(4) &= [1, 0, 0, 0, 0, 0] \\
  C(1) &= [0, 0, 0, 1, 0, 0] \\
  C(2) &= [0, 0, 0, 1, 0, 0] \\
  C(3) &= [0, 0, 0, 0, 0, 1] \\
  C(4) &= [0, 0, 1, 0, 0, 0]
\end{align*}
\]
zk-SNARK: A Simple Example (3)

- The solution S should satisfy all the four constraints
- Can satisfy them in one shot? Quadratic Arithmetic Program (QAP)

- $a = [0, 1, 0, 0, 0, 0]$  
  - $b = [0, 1, 0, 0, 0, 0]$  
  - $c = [0, 0, 0, 1, 0, 0]$  
  - $a = [0, 1, 0, 0, 0, 0]$  
  - $b = [0, 0, 0, 1, 0, 0]$  
  - $c = [0, 1, 0, 0, 1, 0]$  
  - $a = [5, 0, 0, 0, 1]$  
  - $b = [0, 1, 0, 0, 0, 0]$  
  - $c = [0, 0, 0, 0, 1]$  

- $A(x) = [A_1(x), A_2(x), A_3(x), A_4(x), A_5(x), A_6(x)]$  
  - $b = [1, 0, 0, 0, 0, 0]$  
  - $c = [0, 0, 0, 0, 1]$  
  - $a = [5, 0, 0, 0, 1]$  
  - $b = [1, 0, 0, 0, 0, 0]$  
  - $c = [0, 0, 1, 0, 0, 0]$  

- $B(x) = [B_1(x), B_2(x), B_3(x), B_4(x), B_5(x), B_6(x)]$  
  - $A(1) = [0, 1, 0, 0, 0, 0]$  
  - $A(2) = [0, 0, 0, 1, 0, 0]$  
  - $A(3) = [0, 1, 0, 0, 1, 0]$  
  - $A(4) = [5, 0, 0, 0, 0, 1]$  
  - $B(1) = [0, 1, 0, 0, 0, 0]$  
  - $B(2) = [0, 1, 0, 0, 0, 0]$  
  - $B(3) = [1, 0, 0, 0, 0, 0]$  

- $C(x) = [C_1(x), C_2(x), C_3(x), C_4(x), C_5(x), C_6(x)]$  
  - $A(1) = [0, 1, 0, 0, 0, 0]$  
  - $A(2) = [0, 0, 0, 1, 0, 0]$  
  - $A(3) = [0, 1, 0, 0, 1, 0]$  
  - $A(4) = [5, 0, 0, 0, 0, 1]$  
  - $B(1) = [0, 1, 0, 0, 0, 0]$  
  - $B(2) = [0, 1, 0, 0, 0, 0]$  
  - $B(3) = [1, 0, 0, 0, 0, 0]$  

- $A(x) = [0, 1, 0, 0, 0, 0]$  
  - $A(2) = [0, 0, 0, 1, 0, 0]$  
  - $A(3) = [0, 1, 0, 0, 1, 0]$  
  - $A(4) = [5, 0, 0, 0, 0, 1]$  
  - $B(1) = [0, 1, 0, 0, 0, 0]$  
  - $B(2) = [0, 1, 0, 0, 0, 0]$  
  - $B(3) = [1, 0, 0, 0, 0, 0]$  

- $B(x) = [0, 0, 0, 1, 0, 0]$  
  - $C(1) = [0, 0, 0, 1, 0, 0]$  
  - $C(2) = [0, 0, 0, 1, 0, 0]$  
  - $C(3) = [0, 0, 0, 0, 1]$  
  - $C(4) = [0, 0, 1, 0, 0, 0]$  

- $C(x) = [0, 0, 1, 0, 0, 0]$  
  - $C(1) = [0, 0, 0, 1, 0, 0]$  
  - $C(2) = [0, 0, 0, 1, 0, 0]$  
  - $C(3) = [0, 0, 0, 0, 1]$  
  - $C(4) = [0, 0, 1, 0, 0, 0]$
zk-SNARK: A Simple Example (4)

- The final QAP by Lagrange interpolation

- A polynomials
  - \([-5.0, 9.166, -5.0, 0.833]\) = \(-5 + 9.166x - 5x^2 + 0.833x^3\)
  - \([8.0, -11.333, 5.0, -0.666]\) = \(8 -11.333x + 5x^2 -0.666x^3\)
  - \([0.0, 0.0, 0.0, 0.0]\)
  - \([-6.0, 9.5, -4.0, 0.5]\)
  - \([4.0, -7.0, 3.5, -0.5]\)
  - \([-1.0, 1.833, -1.0, 0.166]\)

- B polynomials
  - \([3.0, -5.166, 2.5, -0.333]\)
  - \([-2.0, 5.166, -2.5, 0.333]\)
  - \([0.0, 0.0, 0.0, 0.0]\)
  - \([0.0, 0.0, 0.0, 0.0]\)
  - \([0.0, 0.0, 0.0, 0.0]\)
  - \([0.0, 0.0, 0.0, 0.0]\)

- C polynomials
  - \([0.0, 0.0, 0.0, 0.0]\)
  - \([0.0, 0.0, 0.0, 0.0]\)
  - \([-1.0, 1.833, -1.0, 0.166]\)
  - \([4.0, -4.333, 1.5, -0.166]\)
  - \([-6.0, 9.5, -4.0, 0.5]\)
  - \([4.0, -7.0, 3.5, -0.5]\)

- A(1) =\([0, 1, 0, 0, 0, 0]\)
  - A(2) =\([0, 0, 0, 1, 0, 0]\)
  - A(3) =\([0, 1, 0, 0, 1, 0]\)
  - A(4) =\([5, 0, 0, 0, 0, 1]\)

- B(1) =\([0, 1, 0, 0, 0, 0]\)
  - B(2) =\([0, 1, 0, 0, 0, 0]\)
  - B(3) =\([1, 0, 0, 0, 0, 0]\)
  - B(4) =\([1, 0, 0, 0, 0, 0]\)

- C(1) =\([0, 0, 0, 1, 0, 0]\)
  - C(2) =\([0, 0, 0, 1, 0]\)
  - C(3) =\([0, 0, 0, 0, 1]\)
  - C(4) =\([0, 0, 1, 0, 0]\)
zk-SNARK: A Simple Example (5)

• So where are we?
• We want to prove we know a secret $x$ satisfying an equation: $x^3 + x + 5 = y$
• We convert it into a circuit
• The circuit is converted into a set of R1CS $(a, b, c)$
• R1CS is then transformed to QAP $(A(x), B(x), C(x))$
• Now a solution $s$ should satisfy:
  
  $s \cdot A(x) \ast s \cdot B(x) - s \cdot C(x) = H(x) \ast (x-1)(x-2)(x-3)(x-4) = H(x) \ast Z(x)$
  
  • $s \cdot A(1) \ast s \cdot B(1) - s \cdot C(1) = 0$
  • $s \cdot A(2) \ast s \cdot B(2) - s \cdot C(2) = 0$
  • $s \cdot A(3) \ast s \cdot B(3) - s \cdot C(3) = 0$
  • $s \cdot A(4) \ast s \cdot B(4) - s \cdot C(4) = 0$

• When $x=1,2,3,4$, both sides evaluate to 0
zk-SNARK: A Simple Example (6)

• Solution $s = [1, x, y, y_1, y_2, y_3] = [1, 3, 35, 9, 27, 30]$
• $A \cdot s = [43.0, -73.333, 38.5, -5.166]$
• $B \cdot s = [-3.0, 10.333, -5.0, 0.666]$
• $C \cdot s = [-41.0, 71.666, -24.5, 2.833]$

• $t = A \cdot s \cdot B \cdot s - C \cdot s =$
  • $[-88.0, 592.666, -1063.777, 805.833, -294.777, 51.5, -3.444]$
• $Z = (x-1)(x-2)(x-3)(x-4) \rightarrow [24, -50, 35, -10, 1]$
• $h = t / Z = [-3.666, 17.055, -3.444]$
**zk-SNARK: A Simple Example (7)**

- **Note:**
  - $A(x)$, $B(x)$, $C(x)$ are determined by the equation we want to prove for some $x$
  - $s \cdot A(x) \cdot s \cdot B(x) \neq s^2 \cdot A(x) \cdot B(x)$
  - $s \cdot A(x) \cdot s \cdot B(x) - s \cdot C(x) = H(x) \cdot (x-1)(x-2)(x-3)(x-4)$
    - $A(x)$, $B(x)$, $C(x)$ are known
    - $H(x)$ is determined by $s$

- **Prove equality at some secret point $\alpha$ (no one knows it!)**
  - $s \cdot A(\alpha) \cdot s \cdot B(\alpha) - s \cdot C(\alpha) = H(\alpha) \cdot (\alpha - 1)(\alpha - 2)(\alpha - 3)(\alpha - 4) = H(\alpha) \cdot Z(\alpha)$
zk-SNARK: why it works?

• Constraints enforced by the polynomials
  • Polynomials (of R1CS) evaluated at different points (1, 2, 3, 4) corresponds to different constraint
  • If a solution satisfies R1CS specified by the polynomials, then the solution must satisfy all constraints specified by the R1CS

• Verify equality at a secret point \( \alpha \) instead of polynomials
  • If no one knows \( \alpha \), and \( s \cdot A(\alpha) \cdot s \cdot B(\alpha) - s \cdot C(\alpha) = H(\alpha) \cdot Z(\alpha) \), then \( s \cdot A(x) \cdot s \cdot B(x) - s \cdot C(x) = H(x) \cdot Z(x) \) for any \( x \).
  • Evaluation of \( A(\alpha), B(\alpha), C(\alpha) \) and \( H(\alpha) \) at secret \( \alpha \)

• Need to protect \( s \) and \( \alpha \)
zk-SNARK: A Simple Example (8)

• Prover knows a solution $s$, wanting to prove it satisfies the constraints specified by QAP
  • E.g. a ZeroCash transaction should satisfy constraints on input values, output values.

• $s \cdot A(x) \cdot s \cdot B(x) - s \cdot C(x) = H(x) \cdot (x-1)(x-2)(x-3)(x-4)$

• Use discrete log to hide $s$

• Recall: $y = g^x \mod p$, infeasible to obtain $x$ with $y$ and $p$, where $p$ is a large prime.

• That is: $g^s \cdot A(x) \cdot s \cdot B(x) - s \cdot C(x) = g^{H(x) \cdot (x-1)(x-2)(x-3)(x-4)}$?

• Not that simple!
zk-SNARK: Evaluation at a secret point $\alpha$

- $g^s \cdot A(\alpha) * s \cdot B(\alpha) - s \cdot C(\alpha) = g^{H(\alpha) * (\alpha - 1)(\alpha - 2)(\alpha - 3)(\alpha - 4)}$

- $g^P(\alpha) = g^{a_0 + a_1 \alpha + \cdots + a_n \alpha^n} = g^{a_0 (g^\alpha)^{a_1} \ldots (g^\alpha)^{a_n}}$
  - Recall $A(\alpha) = [A_1(\alpha), A_2(\alpha), \ldots, A_6(\alpha)]$
  - Easy to compute $g^{A_i(\alpha)}$ as $A_i(\alpha)$ is fixed for a circuit

- $s = [1, x, y, y_1, y_2, y_3] = [c_1, c_2, c_3, c_4, c_5, c_6]$, a given solution.

- $g^{s \cdot A(\alpha)} = g^{\sum c_i A_i(\alpha)} = \prod (g^{A_i(\alpha)})^{c_i}$ without leaking $c_i$

- Same way to compute $g^{s \cdot B(\alpha)}, g^{s \cdot C(\alpha)}, g^{H(\alpha)}, g^{(\alpha - 1)(\alpha - 2)(\alpha - 3)(\alpha - 4)}$
  - Public key: $g^{A_i(\alpha)}, g^{B_i(\alpha)}, g^{C_i(\alpha)}, g^{\alpha}, \ldots, g^{\alpha^k}$
  - $H(\alpha)$ depends on $s$, the given inputs and outputs
zk-SNARK: A Summary

• Step 1: Given a set of arithmetic equations to be verified:
  • $H(x) = y$
  • $x + y = z$

• Step 2: Convert equations to R1CS

• Step 3: Convert R1CS to QAP by Lagrange interpolation

• Step 4: Convert to zk-SNARK
Zerocash: Improvement Over Zerocoin (IEEE S&P 2014)

• Zk-SNARKs: zero-knowledge Succinct Non-interactive Arguments of Knowledge.

• Overview:
  • **Mint** coins: from Bitcoin to Zerocash.
  • **Pour**: from two inputs to two new addresses, where zk-SNARK is used.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time (Intel Core i7-4770 @ 3.40GHz with 16GB of RAM (1 thread))</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Setup</strong></td>
<td></td>
</tr>
<tr>
<td>pp</td>
<td>896 MiB</td>
</tr>
<tr>
<td><strong>CreateAddress</strong></td>
<td></td>
</tr>
<tr>
<td>Time</td>
<td>326.0 ms</td>
</tr>
<tr>
<td>addr&lt;sub&gt;pk&lt;/sub&gt;</td>
<td>343 B</td>
</tr>
<tr>
<td>addr&lt;sub&gt;sk&lt;/sub&gt;</td>
<td>319 B</td>
</tr>
<tr>
<td><strong>Mint</strong></td>
<td></td>
</tr>
<tr>
<td>Time</td>
<td>23 μs</td>
</tr>
<tr>
<td>Coin c</td>
<td>463 B</td>
</tr>
<tr>
<td>tx&lt;sub&gt;Mint&lt;/sub&gt;</td>
<td>72 B</td>
</tr>
<tr>
<td><strong>Pour</strong></td>
<td></td>
</tr>
<tr>
<td>tx&lt;sub&gt;Pour&lt;/sub&gt;</td>
<td>996 B&lt;sup&gt;16&lt;/sup&gt;</td>
</tr>
<tr>
<td><strong>VerifyTransaction</strong></td>
<td></td>
</tr>
<tr>
<td>mint</td>
<td>8.3 μs</td>
</tr>
<tr>
<td>pour (excludes &lt;i&gt;L&lt;/i&gt; scan)</td>
<td>5.7 ms</td>
</tr>
<tr>
<td><strong>Receive</strong></td>
<td></td>
</tr>
<tr>
<td>Time (per pour tx)</td>
<td>1.6 ms</td>
</tr>
</tbody>
</table>
Basic anonymous e-cash [Sander Ta-Shma 1999]

Minting:
I hereby spend 1 BTC to create cm

Spending:
I'm using up a coin with (unique) sn, and here are its cm and r.

Legend:
- In private wallet
- In public ledger
- Proved to be known
Basic anonymous e-cash [Sander Ta-Shma 1999]

Minting:
I hereby spend 1 BTC to create cm

Spending:
I’m using up a coin with (unique) sn, and I know r, and a cm in the tree with root, that match sn.

Legend:
- In private wallet
- In public ledger
- Proved to be known
Basic anonymous e-cash – requisite proofs

Spending:
I’m using up a coin with (unique) sn, and I know a cm in the tree, and r, that match sn.

Requires:
zero knowledge
succinct
noninteractive
argument
of knowledge

zkSNARK
I hereby spend $v$ BTC to create $cm$, and here is $k, r'$ to prove consistency.

I’m using up a coin with value $v$ (unique) $sn$, and I know $r', r''$ that are consistent with $cm$.  

**Adding variable denomination**

Minting:

```
i hereby spend \( v \) BTC to create \( cm \),
and here is \( k, r' \) to prove consistency.
```

Spending:

```
i’m using up a coin with value \( v \) (unique) \( sn \), and
I know \( r', r'' \) that are consistent with \( cm \).
```
Pouring Zerocash coins

Single transaction type capturing:

- Sending payments
- Making change
- Exchanging into bitcoins
- Transaction fees

Pour old Zerocash coin old Zerocash coin

\[ \downarrow v_1 \downarrow v_2 \downarrow \text{dest}_1 \downarrow \text{dest}_2 \downarrow v_{\text{pub}} \]

<table>
<thead>
<tr>
<th>old Zerocash coin</th>
<th>new Zerocash coin</th>
<th>new Zerocash coin</th>
<th>new Zerocash coin</th>
</tr>
</thead>
<tbody>
<tr>
<td>( v_1 ) to \text{dest}_1</td>
<td>( v_2 ) to \text{dest}_2</td>
<td>public bitcoins of value ( v_{\text{pub}} )</td>
<td></td>
</tr>
</tbody>
</table>

\[ \downarrow \text{sn}_1 \downarrow \text{sn}_2 \downarrow \text{cm}_1 \downarrow \text{cm}_2 \cdots \downarrow \text{proof} \]

the old coins were \textbf{valid}, and values of old coins \( \sum \)
Pouring Zerocash coins

Single transaction type capturing:

- Sending payments
- Making change
- Exchanging into bitcoins
- Transaction fees

old Zerocash coin → Pour → new Zerocash coin

old Zerocash coin → Pour → new Zerocash coin

The old coins were valid, and values of old coins = \( v_1 + v_2 + v_{\text{pub}} \)
What zk-SNARK can do?

• Inputs equal to outputs
• Hash value matches without disclosing preimage
• Inputs are valid and not double spending

\[ \text{tx}_{\text{pour}} = (rt, s_{n_1}^{\text{old}}, s_{n_2}^{\text{old}}, cm_1^{\text{new}}, cm_2^{\text{new}}, v_{\text{pub}}, \text{info}, \ast) \]

(a) The coin commitment \( cm_i^{\text{old}} \) of \( c_i^{\text{old}} \) appears on the ledger, i.e., \( \text{path}_i \) is a valid authentication path for leaf \( cm_i^{\text{old}} \) with respect to root \( rt \), in a CRH-based Merkle tree.
(b) The address secret key \( a_{sk,i}^{\text{old}} \) matches the address public key of \( c_i^{\text{old}} \), i.e., \( a_{pk,i}^{\text{old}} = \text{PRF}_{a_{sk,i}^{\text{old}}}^{\text{addr}}(0) \).
(c) The serial number \( s_{n_i}^{\text{old}} \) of \( c_i^{\text{old}} \) is computed correctly, i.e., \( s_{n_i}^{\text{old}} = \text{PRF}_{a_{sk,i}^{\text{old}}}^{s_{n_i}^{\text{old}}}(\rho_i^{\text{old}}) \).
(d) The coin \( c_i^{\text{old}} \) is well-formed, i.e., \( cm_i^{\text{old}} = \text{COMM}_{s_i}^{\text{old}}(\text{COMM}_{r_i}^{\text{old}}(a_{pk,i}^{\text{old}} || \rho_i^{\text{old}}) || v_i^{\text{old}}) \).
(e) The coin \( c_i^{\text{new}} \) is well-formed, i.e., \( cm_i^{\text{new}} = \text{COMM}_{s_i}^{\text{new}}(\text{COMM}_{r_i}^{\text{new}}(a_{pk,i}^{\text{new}} || \rho_i^{\text{new}}) || v_i^{\text{new}}) \).
(f) The address secret key \( a_{sk,i}^{\text{old}} \) ties \( h_{\text{Sig}} \) to \( h_i \), i.e., \( h_i = \text{PRF}_{a_{sk,i}^{\text{old}}}^{pk}(i || h_{\text{Sig}}) \).

2. Balance is preserved: \( v_1^{\text{new}} + v_2^{\text{new}} + v_{\text{pub}} = v_1^{\text{old}} + v_2^{\text{old}} \) (with \( v_1^{\text{old}}, v_2^{\text{old}} \geq 0 \) and \( v_1^{\text{old}} + v_2^{\text{old}} \leq v_{\max} \).
Example of a Zerocash Pour transaction

```
root
1c4ac4a110e863deecaa050dc5e5153f2b7010af9

sn_1
a365e7006565f14342df9096b46cc7f1d2b9949367180fdd8de4090e30bfc

sn_2
6937031dce13facdebe79e82712ffad2e980c911e4ce8ca9b25fc88df73b52

cm_1
a4d015440f9cfae0uc3ca3a38cf04058262d74b60cb14ed6063e047694580103

cm_2
2ca1f83363ac827ba6ae69b53edc8555e6e2c2d0a24f6ed50d4f542dc772

v_pub
00000000000000042

pubkeyHash info
8f9a43f0fe28bef052ec209724bb0e502ff5427

SigPK
2dd489d97cdaaeb006cb649e1699b1ae108d43

Sig
f1d2df294e866ac866fd7b3669c49bdf32b3e15a38359c82f3d2b3342cb4bedcb78ce116bac69e

MAC_1
b8a5917e5a1587a970bc9e3ec5e395240ceb1ef700276ec0fa92d1835cb7629

MAC_2
ade6218b3a17d6093963ec6694b7b2b446l612699d4bcafa85fcbf39fb54603a

ciphertext_1
048070fe125bdaf93ae6a7c08b65adbb2a43846867243c74e80abc5b74df3524a987a2e3ed075d54ae7a53866793eeea5070c4e0895
4ff5d80caca214ce572f42dc6676f0e59d5b1ed68ad33b0c73cf9eac671d8f0126d866b667b319d255d7002da0a2d8aec47f8fd648
057fa823a253d3f52e86ed65ce229db56816e46967bfaf4d2303af7e09d24b8e302777336cb7d8c81d3c786f1547fe0d00c29b63bd
972aad87b3f12a2667f7a575e

ciphertext_2
04931108141b0b05cbab9a922506203254987c8b8604d96985ca52c71a77055b4979a50099ecfca5a359bdf0411983388fa5e840a0d
64816f1d9f386461d217986af98176f47f0c19a2dc1879acebf14bd978624e80ac272063e6b6f7b8c42c6ee01edfbcddbe60eab586
0eaeadecc6b017069c8be2ebe8a8a2fa5e06780a4e2466d72bc3243e873820b2d2e4b954e9216b566c140de79351abf47254d122a35f
17f840156bd7b1feeb942729dc

zkSNARKproof
a4c3cadd6e02e5c15dc8a37ebc51885cf86c5a04bb1c10cbf3ed97b778277fb8adceb240c40acccf28543ed1e1afdcf3c532bc75aafefef9d3975726f2ca829228
6ca8d4f8da21b398c6fa3ca2130b825448585b1ce4c7eac9e57592e1ed233d434a2e768b9deba536594789ef17002b096f7058df611e2b0c2087618c58208e3
6585f00084613f83ff5139d0180ac1182095cde69432287699e76ed7832a5fc5dc30874f92965b882731523e0fa1a5b649e3df2f958d05dca75637a1
298025s06dfebe9cfe8c8c40d61bcf4e77dcb11467b9e61654fb623d3fba9e7c8ad17f08b17992715dfdd431c9451e0b597dc506dad84a9f8475d4be530eb501925
dfd22981a2590a3799523b99a8e50d0eaaeb5306c10be5
```

~1KB total. Less without direct payments and public outputs.
Tutorial Examples

- libsnark lightning circuit
  - (source) https://github.com/wanzhiguo/lightning_circuit

- zksnark-toy
  - (source) https://github.com/wanzhiguo/zksnark-toy

- libsnark tutorial (originally by Howard Wu from UC Berkerly)
  - https://github.com/SadPencil/libsnark-tutorial
  - git clone http://github.com/SadPencil/libsnark-tutorial

- Checkout ZeroCash’s zkSNARK
- SCIPR-lab/libsnark on github.com
Lightning Circuit

• Given $H_1, H_2, X$, prove one knows $(R_1, R_2)$ s.t.
  • $H_1 = \text{sha256}(R_1)$ and
  • $H_2 = \text{sha256}(R_2)$ and
  • $R_1 = R_2 \text{ XOR } X$

• (docker image)
  • make
  • ./test

• Implementation
  • Compose a gadget: l_gadget
  • Write test.cpp to test it
zksnark-toy

- Given a Merkle root $rt$, a leaf node $leaf$, a path from $leaf$ to $rt$, prove one knows $pre\_leaf$ s.t.
  - $leaf = \text{sha256}(pre\_leaf)$
  - $leaf$ is valid w.r.t. $rt$ and path

- (docker image )
  - $\text{chmod } +x \text{ get-libsnnark}$
  - $./\text{get-libsnnark}$
  - $\text{make}$
  - $\text{src/main}$

- Implementation
  - toy-gadget
  - $\text{main.cpp}$ for test
Experiments

• Write simple gadgets based on examples
  • lightning_circuit
  • zksnark-toy

• Test gadgets with concrete instances
  • Compile your code
  • Run the code

• Your tasks:
  • $H(x) = y$
  • $x + y = z$
  • Any other interesting proof?
References

- Monero explorer. Moneroexplorer.com
- Monero Trasaction Size. https://monero.stackexchange.com/questions/5664/size-requirements-for-different-pieces-of-a-monero-transaction
- Explaining SNARKs. https://blog.z.cash/snark-explain1/
- Explaining zkSNARK: https://blockchain.iethpay.com/zero-knowledge-zkSNARKs.html
- lightning_circuit: https://github.com/ebfull/lightning_circuit
- Libsnark: Github.com/SCIPR-lab/libsnark
Thanks for your attention!

Questions?