Copyrighted Material

区块链基础及应用

Chapter 6 比特币和匿名性

苏 明

概览

- 6.1 匿名的基础知识
- 6.2 如何对比特币去匿名化
- 6.3 混币
- 6.4 分布式混币
- 6.5 零币和零钞

6.1 匿名的基础知识

匿名（Anonymity）：无关联性的化名

- 比特币系统中，使用者不需要使用真实的姓名
- 需要使用公钥哈希值作为交易标识
- 一个用户可以随机创建出任意多个比特币地址

6.1 匿名的基础知识

- 比特币具有化名性，但是*不能达到绝对隐私*

- 使用数字货币（如比特币）支付时，在真实的物理世界里容易暴露身份，进而关联到地址，以及其他所有的交易

6.1 匿名的基础知识

无关联性

- 同一个用户的不同地址应该不易关联
- 同一个用户的不同交易应该不易关联
- 一个交易的交易双方应该不易关联

6.1 匿名的基础知识

- 区块链货币中，所有交易都记录在一个公开账本上，也就是说相关交易信息可以*永久追踪*

✓ 希望**能够达到传统银行能够达到的隐私保护级别**，降低公共区块链带来的信息暴露风险

✓ 超越传统银行给我们的隐私保护级别

6.1 匿名的基础知识

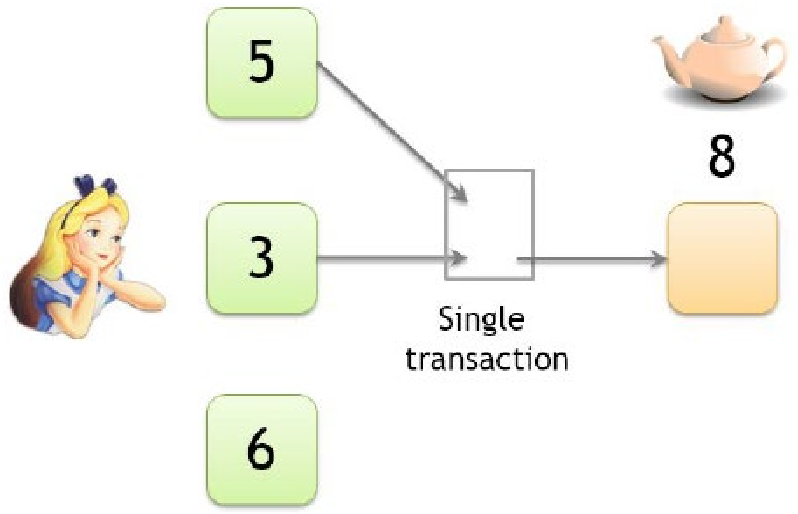
匿名化和去中心化

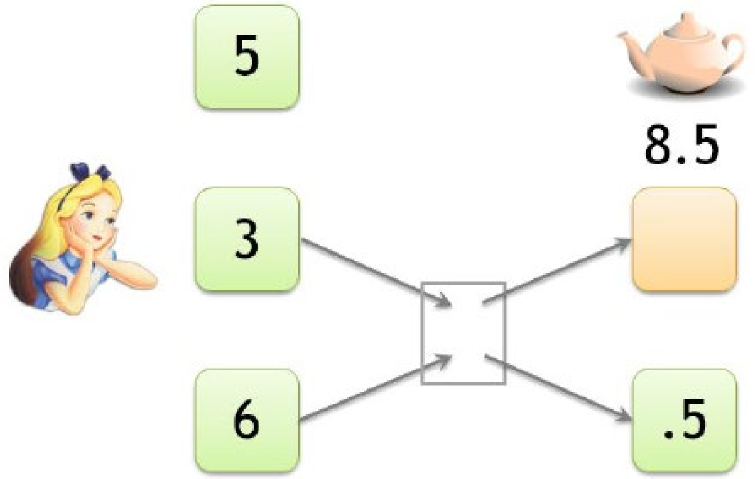
- Chaum的电子现金系统，采用了*盲签名*技术，但还需一个*中央权威机构*

- Zerocoin， Zerocash：匿名化&去中心化的加密数字货币系统

6.2 如何对比特币去匿名化

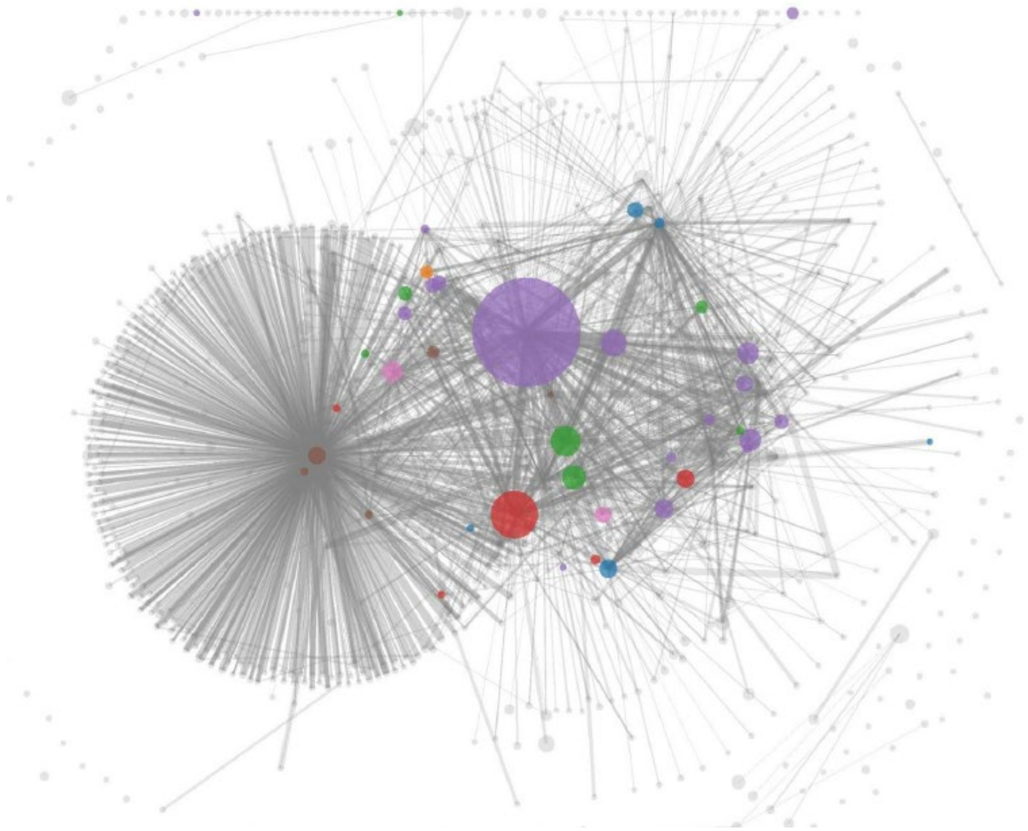
- 关联性





容易暴露零钱地址

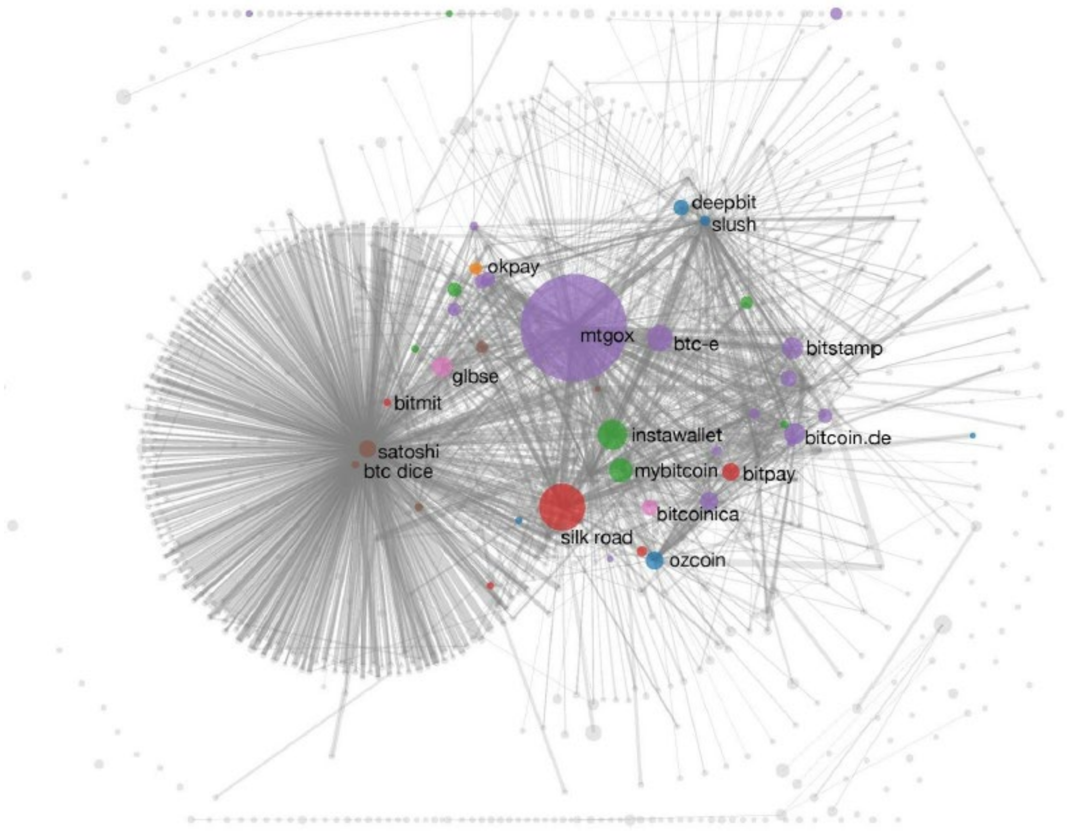
6.2 如何对比特币去匿名化



***Clustering of addresses*：** *Characterizing Payments Among Men with No Names*

6.2 如何对比特币去匿名化

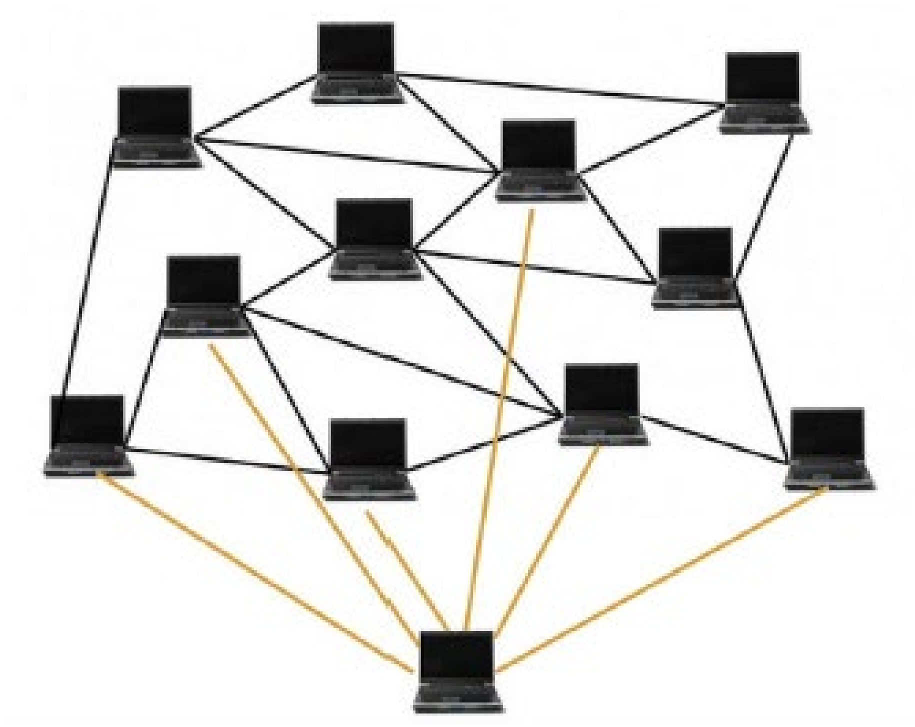
- 利用交易进行标记



交易图谱分析

6.2 如何对比特币去匿名化

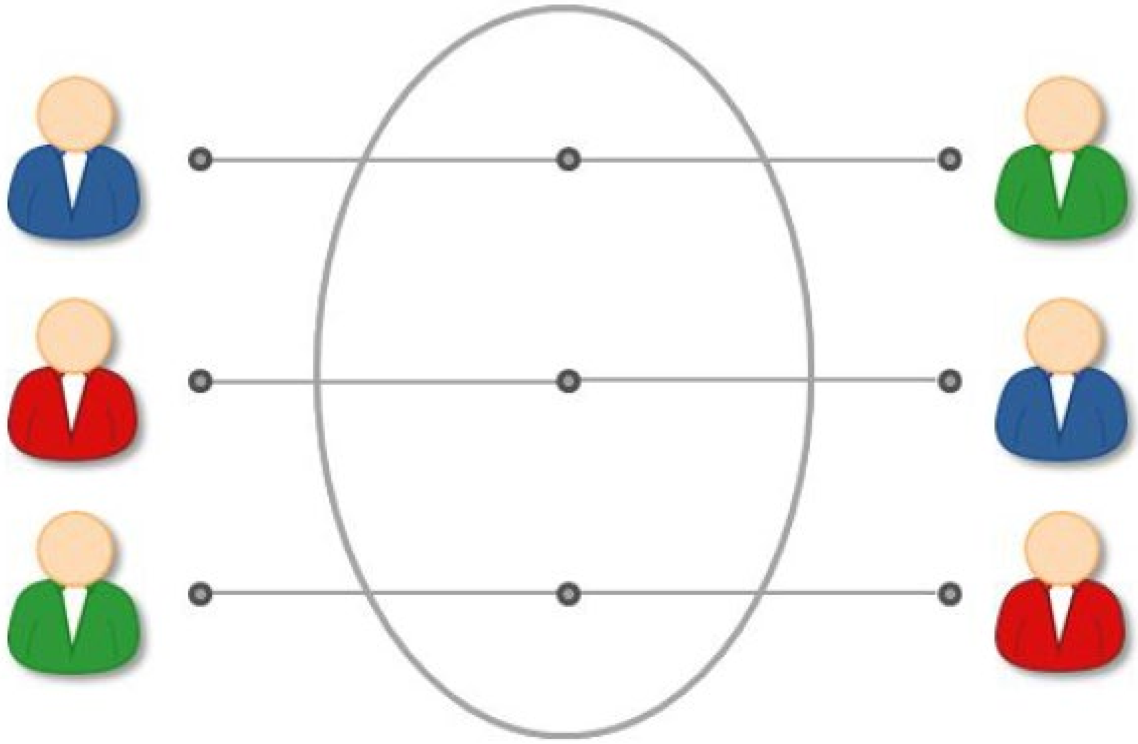
- 网络层的去匿名化



“the first node to inform you of a transaction is probably the source of it.”

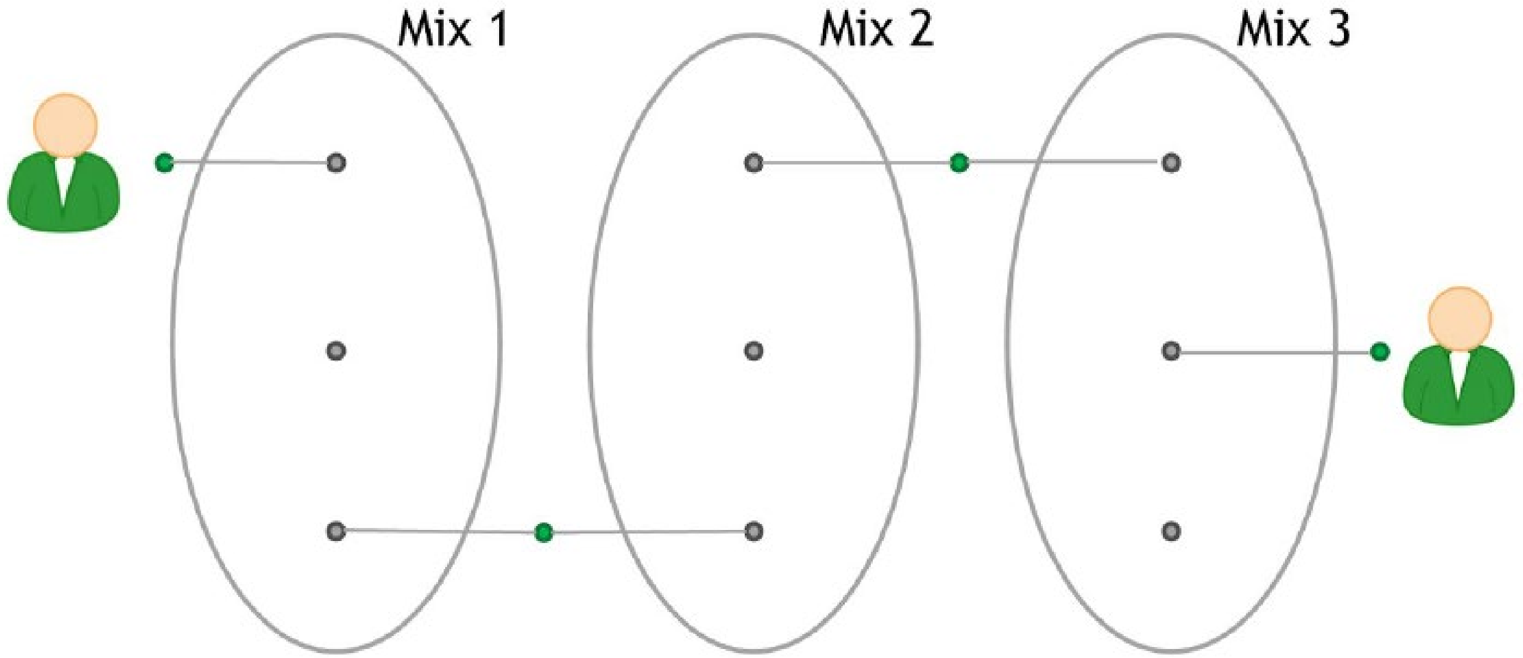
6.3 混币

- 想要匿名化，使用一个中介媒体



6.3 混币

- 多重混币

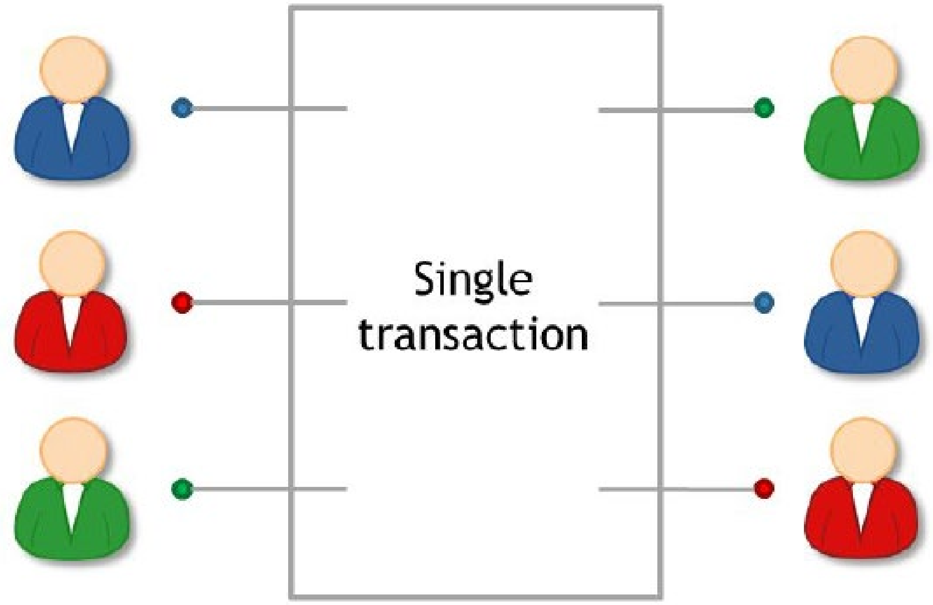


6.4 分布式混币

- 分布式混币(Decentralized Mixing)
- 采用一种用户之间的**点对点**模式实现混币交易的协议

6.4 分布式混币

攻击者无法建立输入和输出的匹配关系



A Coinjoin transaction

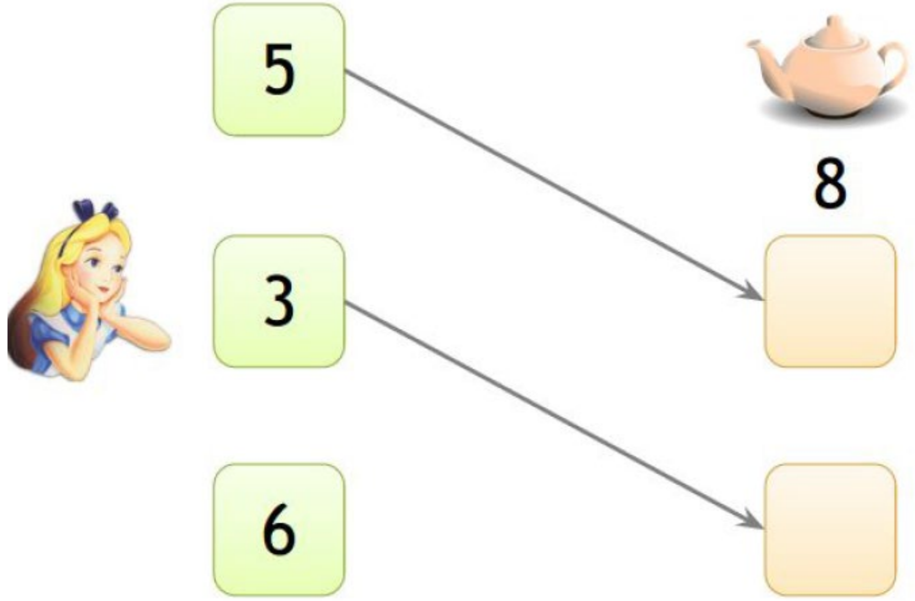
6.4 分布式混币

高交易风险流

- 为了完成一笔支付，用户通常会组合所拥有的数字货币，这样便有足够数额可以支付到**单一**接收地址

- 规避：所有输入地址被关联在一起

6.4 分布式混币



Merge avoidance

零知识证明

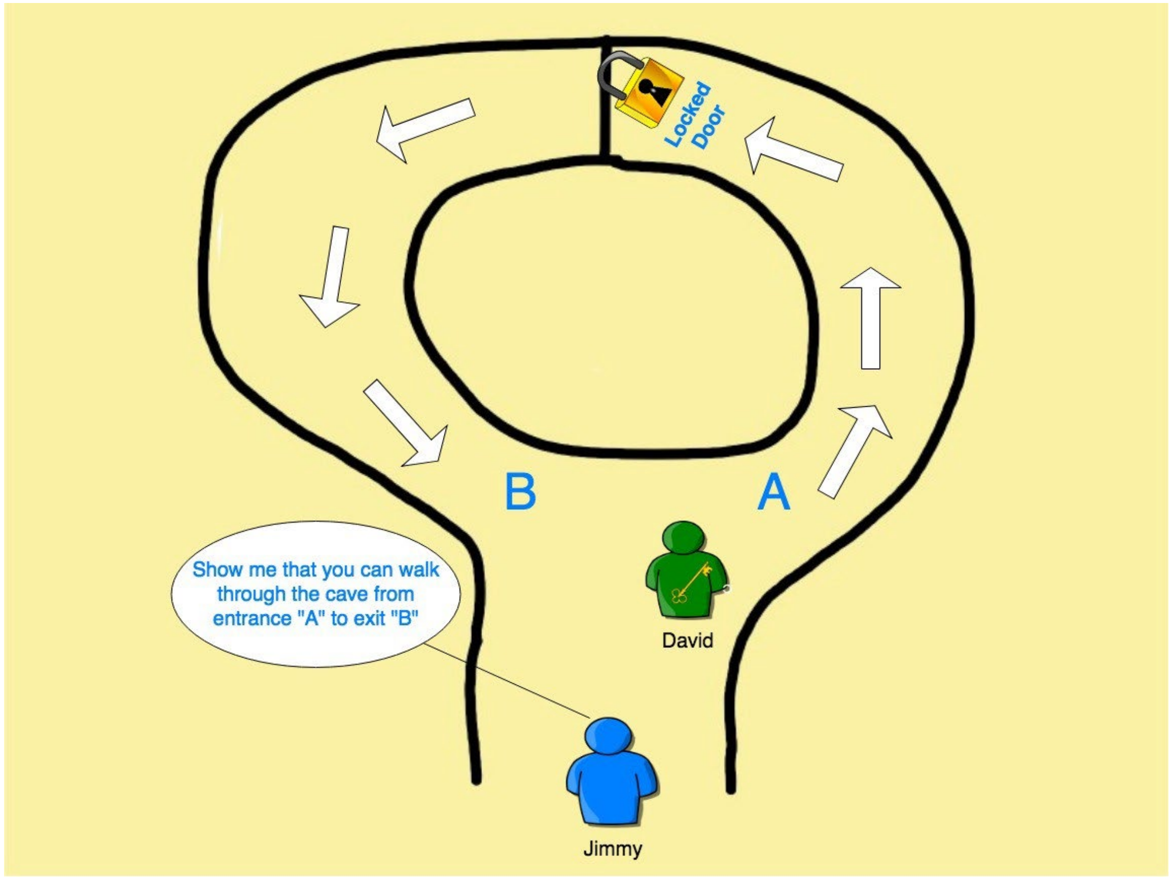
Zero Knowledge Proof：证明者(Prover)要让验证者(Verifier)相信自己拥有某种知识，但又不泄漏它

- 很多场景中有着广泛的应用，比如金融交易中，保护支付方、接收方、交易金额的**隐私**

零知识证明

- Example 1. A Key to a Door
- Example 2. Coloring Problem

Example 1

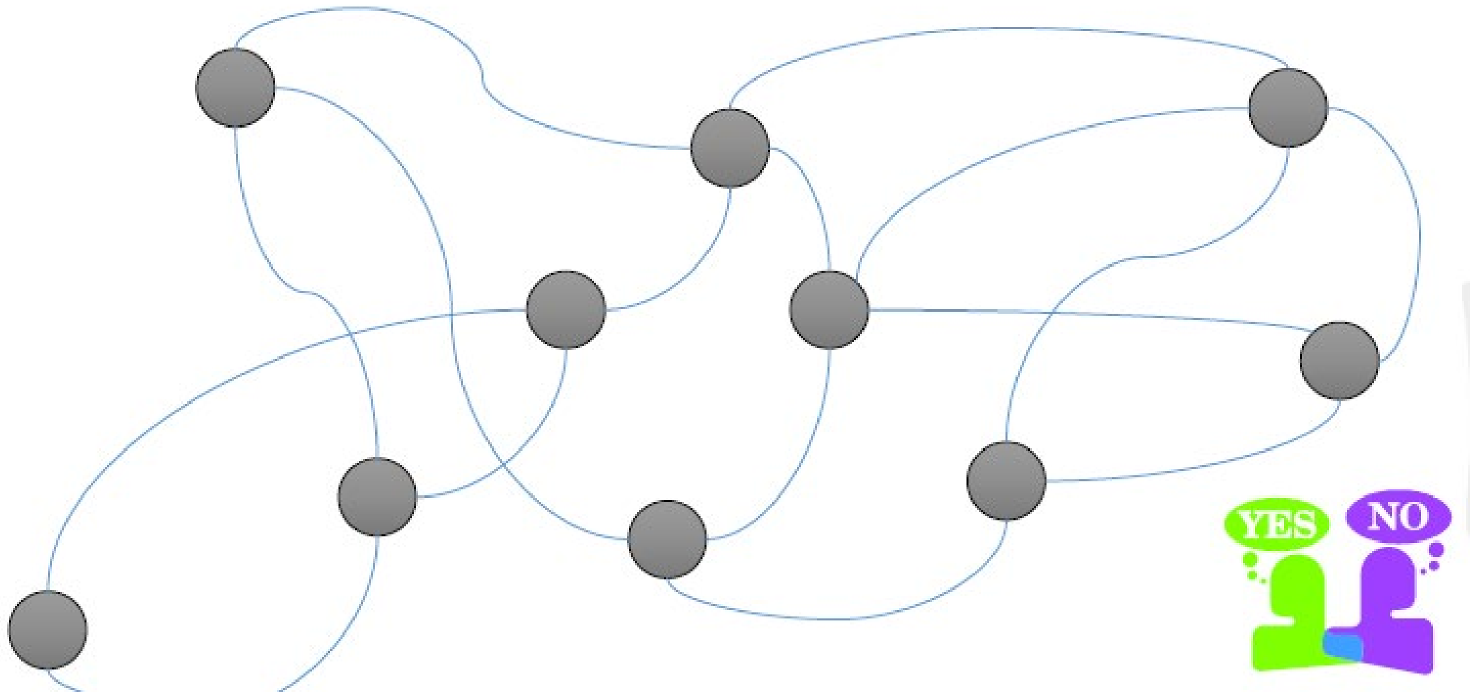

Locked Door
B
A
Show me that you can walk through the cave from entrance "A" to exit "B"
David
Jimmy

Ali Baba cave

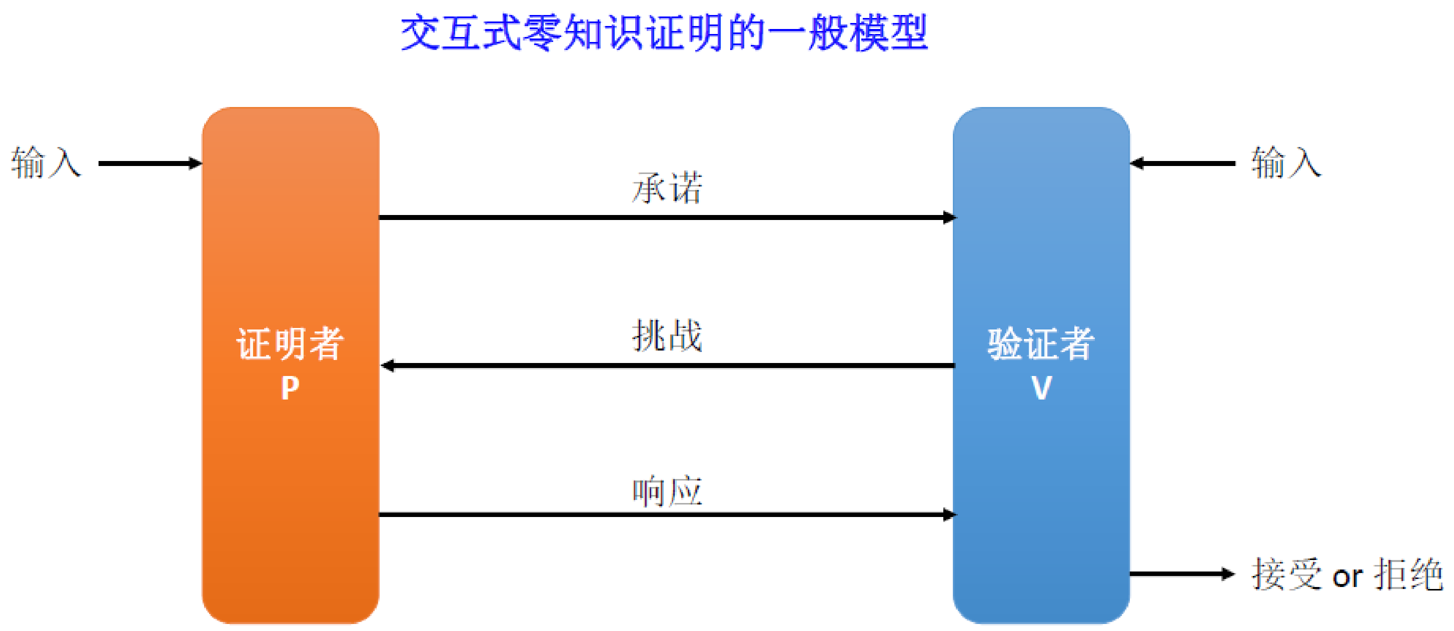
Example 2

Can we label this graph with {r, g, b}?

三色问题



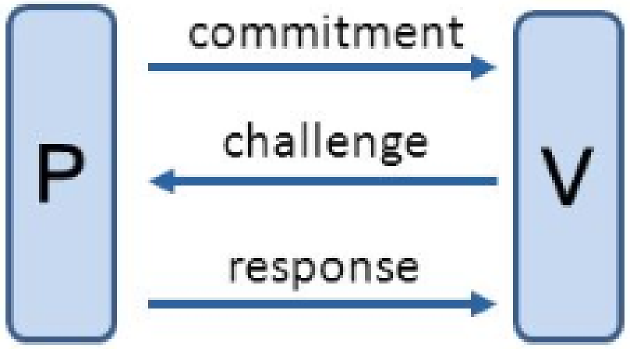
交互式零知识证明的一般模型

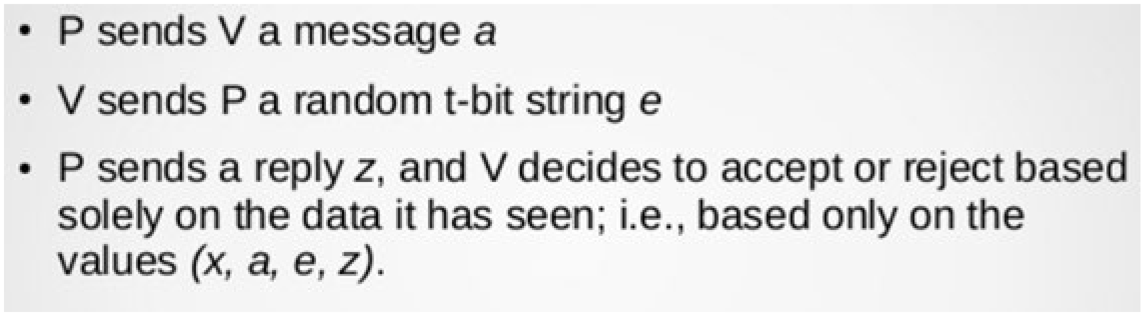


- 证明者和验证者共享一个公共输入，证明者可能拥有某个秘密输入。
- 如果验证者认可证明者的响应，则输入接收（Accept）；否则，输出拒绝（Reject）。

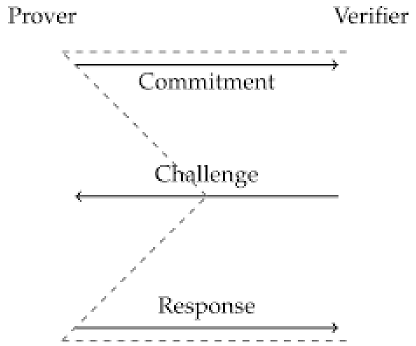
Sigma-Protocol

Sigma-protocols





- P sends V a message *a*
- V sends P a random t-bit string *e*
- P sends a reply *z*, and V decides to accept or reject based solely on the data it has seen; i.e., based only on the values *(x, a, e, z)*.



Zero-Knowledge Proof

A zero-knowledge proof must satisfy three properties:

- **Completeness**: if the statement is true, the honest verifier (that is, one following the protocol properly) will be convinced of this fact by an *honest prover*.
- **Soundness**: if the statement is false, *no cheating prover* can convince the honest verifier that it is true, except with some small probability.
- **Zero-knowledge**: if the statement is true, *no verifier learns anything* other than the fact that the statement is true. In other words, just knowing the statement (not the secret) is sufficient to imagine a scenario showing that the prover knows the secret. This is formalized by showing that every verifier has some *simulator* that, given only the statement to be proved (and no access to the prover), can produce a transcript that "looks like" an interaction between the honest prover and the verifier in question.

Zero-Knowledge Proof

零知识证明

对于语言$L \in \{0,1\}^*$，以及一对交互图灵机$< P, V >$，其中P拥有无限的计算能力，称为证明者，V为概率多项式时间的验证者。

称$< P, V >$为语言L的零知识交互证明系统，如果满足以下条件：

- 完备性（Completeness）：对于任何公共输入$x \in L$，

$$\Pr[(P, V)(x) = 1 | x \in L] \geq 1 - negl(|x|).$$

- 可靠性（Soundness）：对于任意公共输入$x \notin L$和任意无限计算能力的证明者P^*，

$$\Pr[(P^*, V)(x) = 0 | x \notin L] \geq 1 - negl(|x|).$$

- 零知识（Zero-knowledge）：对任意概率多项式时间验证者V^*，都存在一个概率多项式时间的模拟器S，使得任意的$x \in L$，

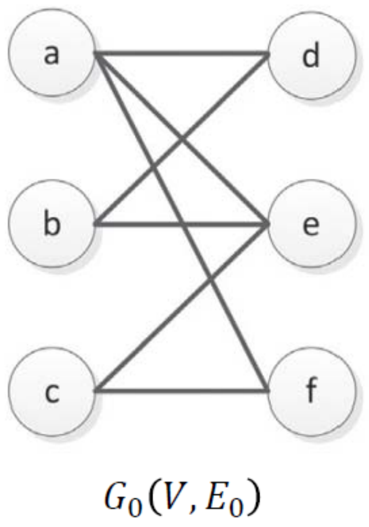
$$< P, V^* > (x) \approx_c S(x).$$

其中，$negl(|x|)$为一个可忽略函数，$\approx_c$ 表示计算不可区分。

交互式零知识证明

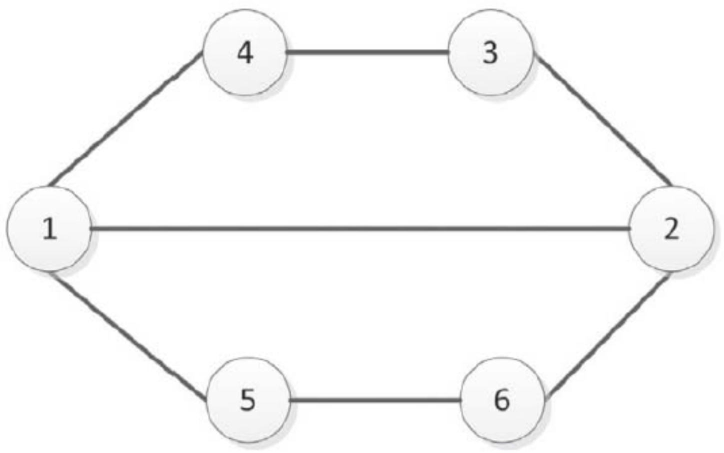
➤ **图同构（Graph Isomorphism）的定义**

Def. 两个图$G_0(V,E_0)$和$G_1\ (V,E_1)$是同构的，当且仅当存在有一个置换$\phi \in S_{|V|}$使得对于任意的$(u,v)\in E_0$仅有$\big(\phi(u),\phi(v)\big)\in E_1$。



$G_0(V,E_0)$

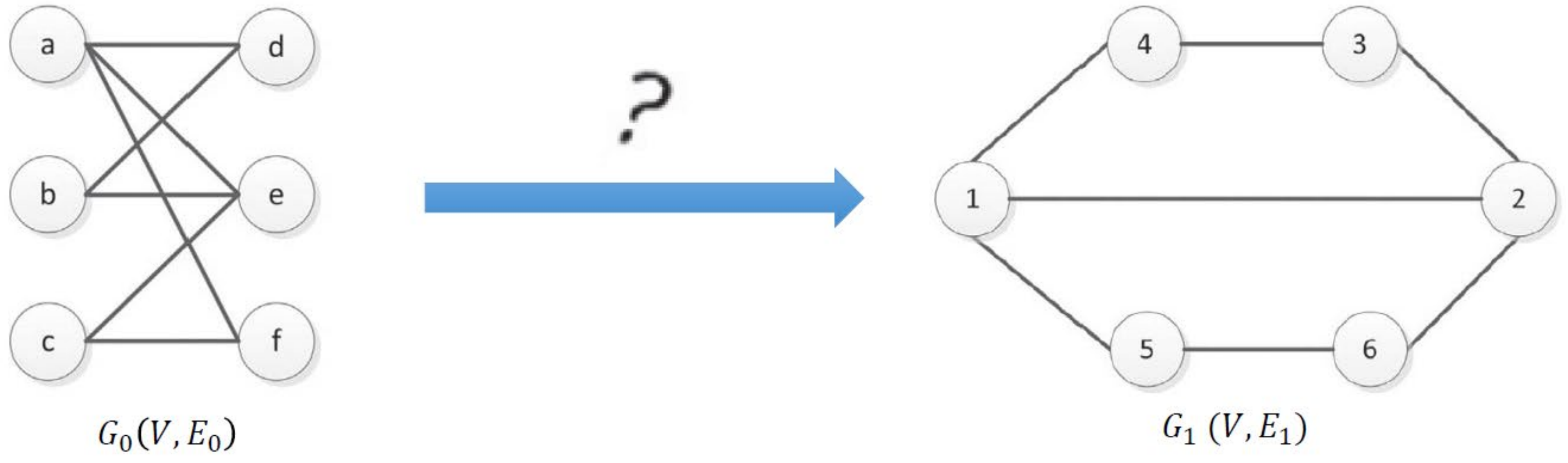
$$\phi:\begin{cases}\phi(a)=1\\ \phi(b)=6\\ \phi(c)=3\\ \phi(d)=5\\ \phi(e)=2\\ \phi(f)=4\end{cases}$$



$G_1\ (V,E_1)$

交互式零知识证明

GI问题：给定两个图G_0和G_1，判断两个图之间是否存在一个同构映射？

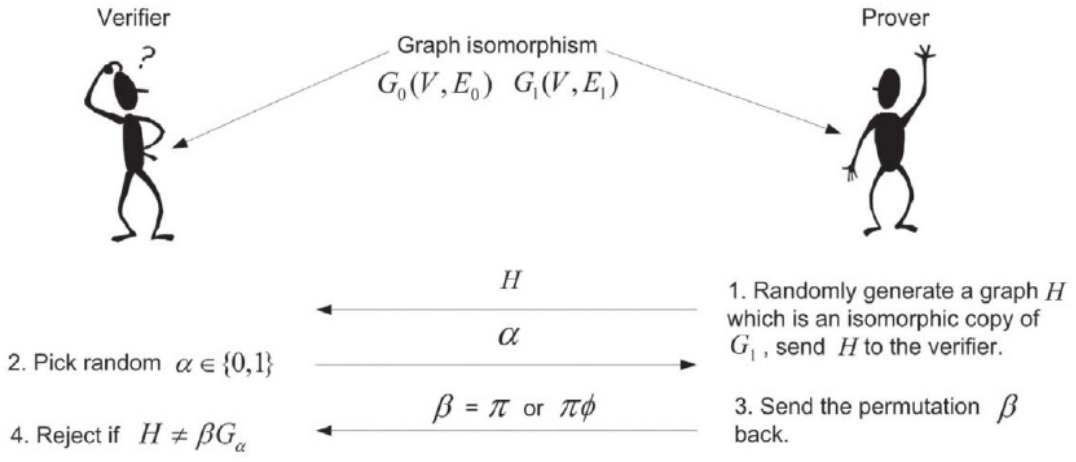


交互式零知识证明

公共输入：两个同构的图$G_0(V, E_0)$和$G_1(V, E_1)$。

1）P: 证明者随机产生一个置换π，并计算图$H = \pi G_1$。然后证明者将H发送给验证者。

2）V: 验证者随机生成一个比特值$\alpha \in_R \{0,1\}$，并将α发送给证明者。


Verifier
Prover
Graph isomorphism
$G_0(V,E_0)$ $G_1(V,E_1)$
H
1. Randomly generate a graph H which is an isomorphic copy of G_1, send H to the verifier.
α
2. Pick random $\alpha \in \{0,1\}$
$\beta = \pi$ or $\pi\phi$
3. Send the permutation β back.
4. Reject if $H \neq \beta G_\alpha$

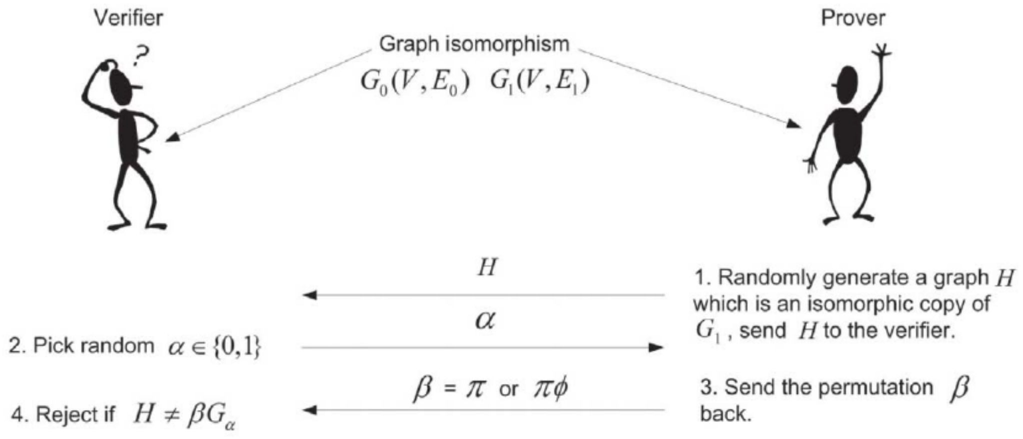
交互式零知识证明

3）P: 证明者根据α做出不同的响应：

- 如果$\alpha = 1$，证明者发送$\beta = \pi$给验证者；
- 如果$\alpha = 0$，证明者发送$\beta = \pi \cdot \phi$给验证者。

4）V: 验证者判断置换β是否是H和G_α的同构。

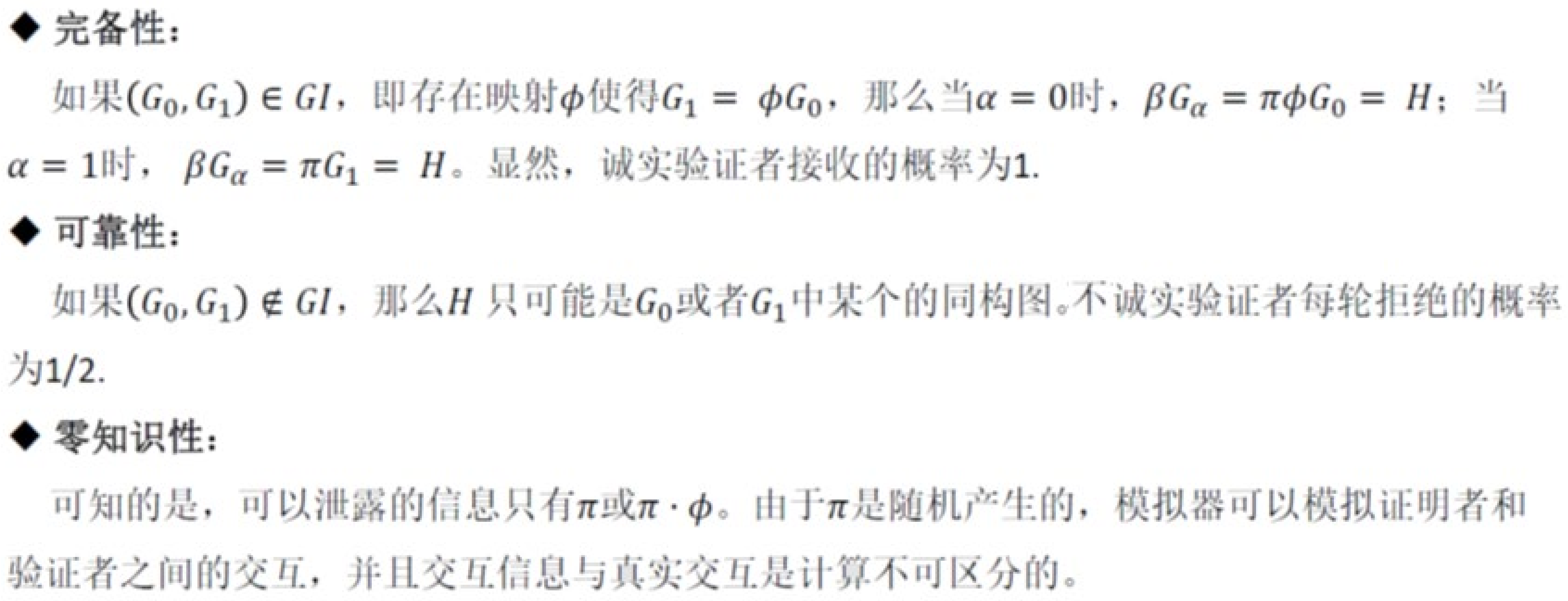
如果不是，验证者拒绝接受证明；否则，验证者进入下一轮证明。


Verifier
Prover
Graph isomorphism
$G_0(V,E_0)$ $G_1(V,E_1)$
H
1. Randomly generate a graph H which is an isomorphic copy of G_1, send H to the verifier.
α
2. Pick random $\alpha \in \{0,1\}$
$\beta = \pi$ or $\pi\phi$
3. Send the permutation β back.
4. Reject if $H \neq \beta G_\alpha$

验证者执行上述证明过程t轮后，且均未拒绝，则验证者接受证明者的证明，即相信两个图是同构的。

交互式零知识证明

图同构问题零知识证明



◆ **完备性：**

如果$(G_0, G_1) \in GI$，即存在映射ϕ使得$G_1 = \phi G_0$，那么当$\alpha = 0$时，$\beta G_\alpha = \pi\phi G_0 = H$；当$\alpha = 1$时，$\beta G_\alpha = \pi G_1 = H$。显然，诚实验证者接收的概率为1.

◆ **可靠性：**

如果$(G_0, G_1) \notin GI$，那么H 只可能是G_0或者G_1中某个的同构图。不诚实验证者每轮拒绝的概率为1/2.

◆ **零知识性：**

可知的是，可以泄露的信息只有π或$\pi \cdot \phi$。由于π是随机产生的，模拟器可以模拟证明者和验证者之间的交互，并且交互信息与真实交互是计算不可区分的。

交互式零知识证明

- Schnorr （身份鉴别）协议

用户注册流程：

用户P随机选取一个ω（$1 \le s \le q-1$）作为秘密值，并计算$h = g^{\omega} mod\ p$。之后，用户P向TA注册并发布公钥h。

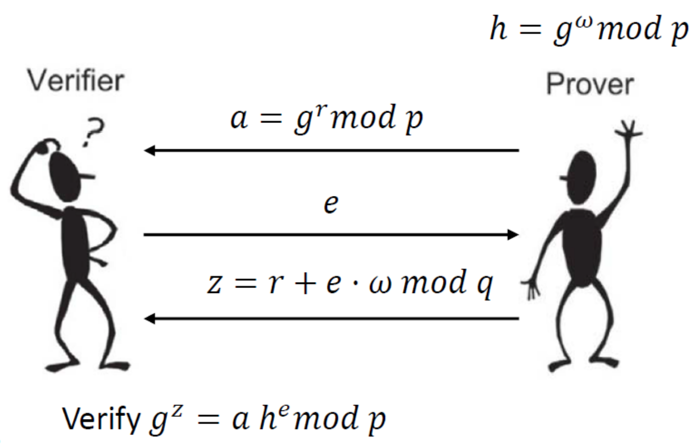
鉴别协议流程：

1）P随机选择r，$1 \le r \le q-1$，计算并发送$a = g^{r} mod\ p$给V；

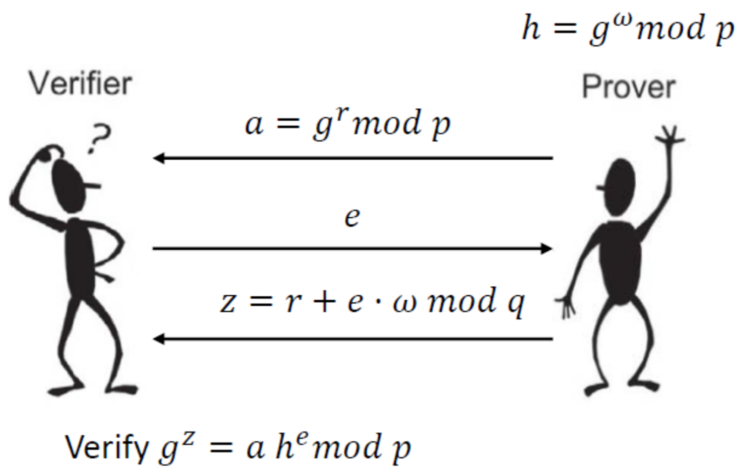
2）V随机选择e，$1 \le e \le 2^{t}$，发送e给P；

3）P计算并发送$z = r + e \cdot \omega\ mod\ q$给V；

4）V验证$g^{z} = a\ h^{e} mod\ p$是否成立。若成立，则V接受；否则，V拒绝。


$h = g^{\omega} mod\ p$
Verifier
Prover
$a = g^{r} mod\ p$
e
$z = r + e \cdot \omega\ mod\ q$
Verify $g^{z} = a\ h^{e} mod\ p$

交互式零知识证明



$h = g^{\omega} mod\ p$

Verifier

Prover

$a = g^{r} mod\ p$

e

$z = r + e \cdot \omega\ mod\ q$

Verify $g^{z} = a\ h^{e} mod\ p$

➢ **协议性质分析**

◆ **完备性：** 显然，诚实验证者接收的概率为1。

◆ **可靠性：** 如果证明者不知道秘密值ω，那么他只能够以2^{-t}的概率欺骗验证者。

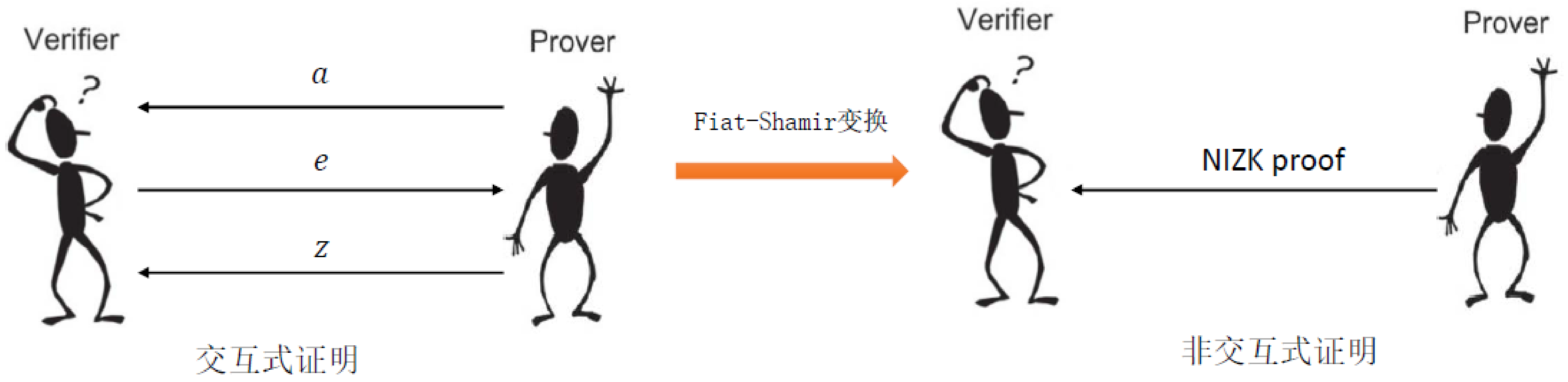
◆ **零知识性：** 可被模拟器模拟，达到计算不可区分性。

如果t比较大时，每轮的错误概率就会非常低，几乎可以忽略不计。

非交互式零知识证明

Fiat-Shamir变换是一种可以将Sigma协议变成非交互证明的技术。它能够让证明者Prover可以通过给验证者Verifier发送一个证明信息即可完成证明(无需交互，无需返回挑战)。

而且，它能把任何一个Simga协议变成一个数字签名，签名的含义就是“知道这个Sigma协议的秘密的人已经签署了这个消息”。Prover能够创造一个证明，然后分发给很多个验证者，验证者可以不必联系Prover即可验证证明有效性。同时零知识也变得容易了，因为验证者或者其他敌手不能做任何事情。



一个密码学安全的 Hash 函数可以近似地模拟传说中的「随机预言机」

6.5 Zerocoin & Zerocash

- Zerocoin:

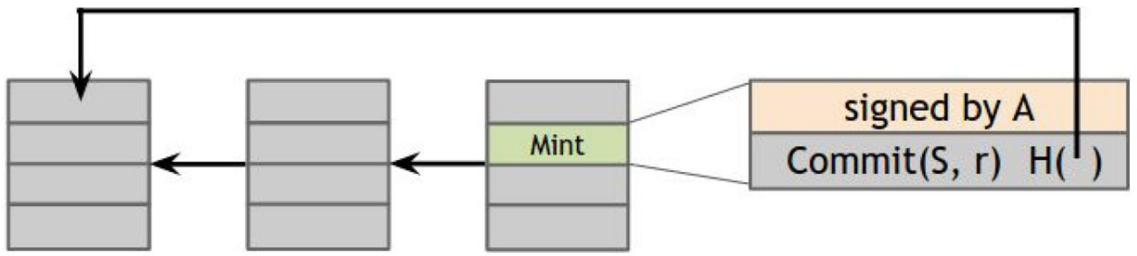
I know x such that `H(x || ⟨other known inputs⟩ ) < ⟨target⟩ .`

"I know x such that H(x) belongs to the following set: {...}".
The proof **would reveal nothing** about x

6.5 Zerocoin & Zerocash

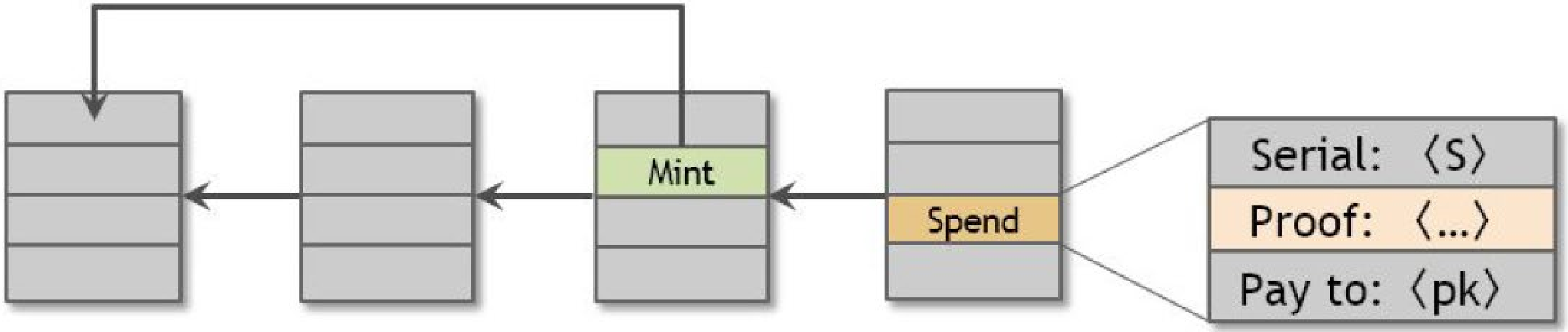
- Zerocoin

Committing to a serial number



Putting a zerocoin on the block chain

6.5 Zerocoin & Zerocash



Spending a zerocoin

6.5 Zerocoin & Zerocash

Spending a zerocoin with serial number S to redeem a new basecoin

- Create a special "spend" transaction that contains S, along with a zero-knowledge proof of the statement:

  "I know *r* such that *Commit(S, r)* is in the set $\{c_1, c_2, \ldots, c_n\}$".
- Miners will verify your zero-knowledge proof which establishes your *ability* to open one of the zerocoin commitments on the block chain, without actually opening it.
- Miners will also check that the serial number S has never been used in any previous spend transaction (since that would be a double-spend).
- The output of your spend transaction will now act as a new basecoin. For the output address, you should use an address that you own.

6.5 Zerocoin & Zerocash

Zerocoin：匿名性

- 铸币交易或者花费交易中***没有展示过r***
- ***无人知道序列号对应***哪一个具体的零币

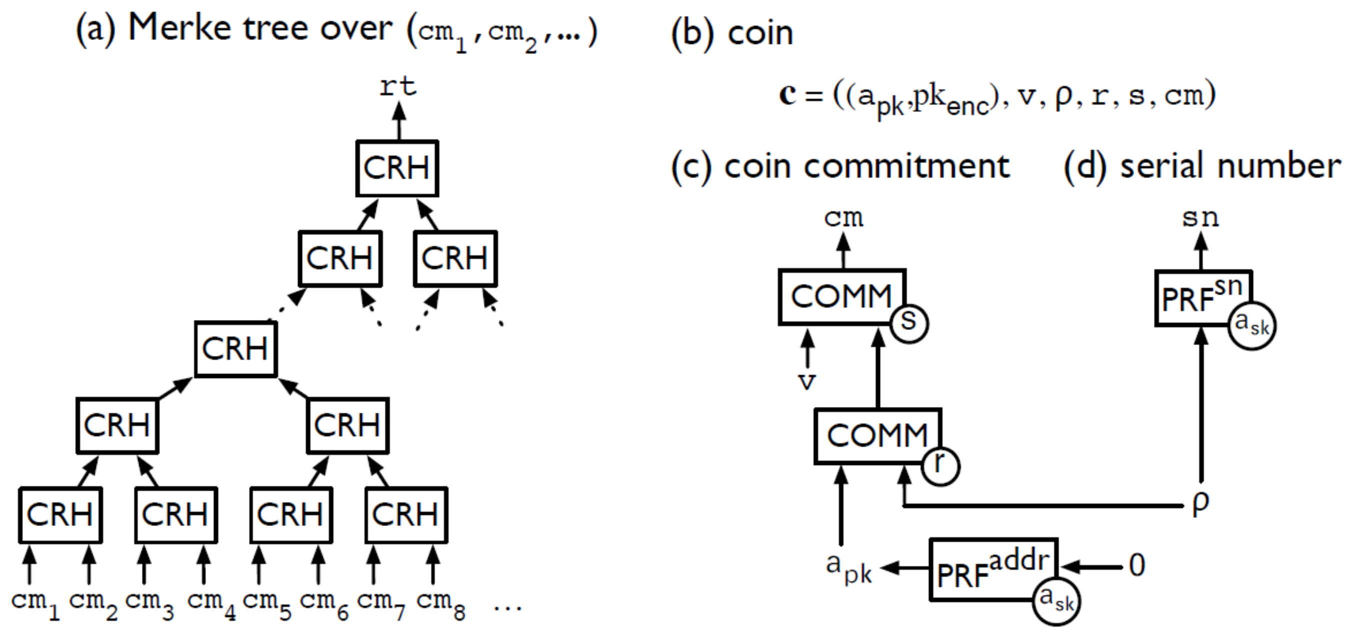
6.5 Zerocoin & Zerocash

Zerocash

- zk-SNARK (Zeroknowledge Succinct Non-interactive Arguments of Knowledge)
- **DAP** (Decentralized Anonymous Payment Scheme)
- *Hiding user identities, transaction amounts, and account balances from public view*

6.5 Zerocoin & Zerocash

Zerocash



(a) Merke tree over (cm₁, cm₂, ...)
rt
CRH
cm₁ cm₂ cm₃ cm₄ cm₅ cm₆ cm₇ cm₈ ...
(b) coin
c = ((a_pk, pk_enc), v, ρ, r, s, cm)
(c) coin commitment
(d) serial number
cm
COMM
s
v
r
a_pk
PRF^addr
a_sk
0
ρ
sn
PRF^sn
a_sk

$\mathtt{rt}$	=	*Merkle-tree root*
$\mathtt{cm}$	=	*coin commitment*
$\mathtt{sn}$	=	*serial number*
$\mathtt{v}$	=	*coin value*
$\mathtt{r}, \mathtt{s}$	=	*commitment rand.*
ρ	=	*serial number rand.*
(a_{pk}, pk_{enc})	=	*address public key*
(a_{sk}, sk_{enc})	=	*address secret key*

6.5 Zerocoin & Zerocash

Zerocash

CreateAddress

- INPUTS: public parameters pp
- OUTPUTS: address key pair $(\mathsf{addr_{pk}}, \mathsf{addr_{sk}})$

1) Compute $(\mathsf{pk_{enc}}, \mathsf{sk_{enc}}) := \mathcal{K}_{\mathsf{enc}}(\mathsf{pp_{enc}})$.
2) Randomly sample a $\mathsf{PRF}^{\mathrm{addr}}$ seed a_{sk}.
3) Compute $a_{\mathsf{pk}} = \mathsf{PRF}^{\mathrm{addr}}_{a_{\mathsf{sk}}}(0)$.
4) Set $\mathsf{addr_{pk}} := (a_{\mathsf{pk}}, \mathsf{pk_{enc}})$.
5) Set $\mathsf{addr_{sk}} := (a_{\mathsf{sk}}, \mathsf{sk_{enc}})$.
6) Output $(\mathsf{addr_{pk}}, \mathsf{addr_{sk}})$.

Mint

- INPUTS:
  - public parameters pp
  - coin value $v \in \{0, 1, \dots, v_{\mathsf{max}}\}$
  - destination address public key $\mathsf{addr_{pk}}$
- OUTPUTS: coin $\mathbf{c}$ and mint transaction $\mathsf{tx_{Mint}}$

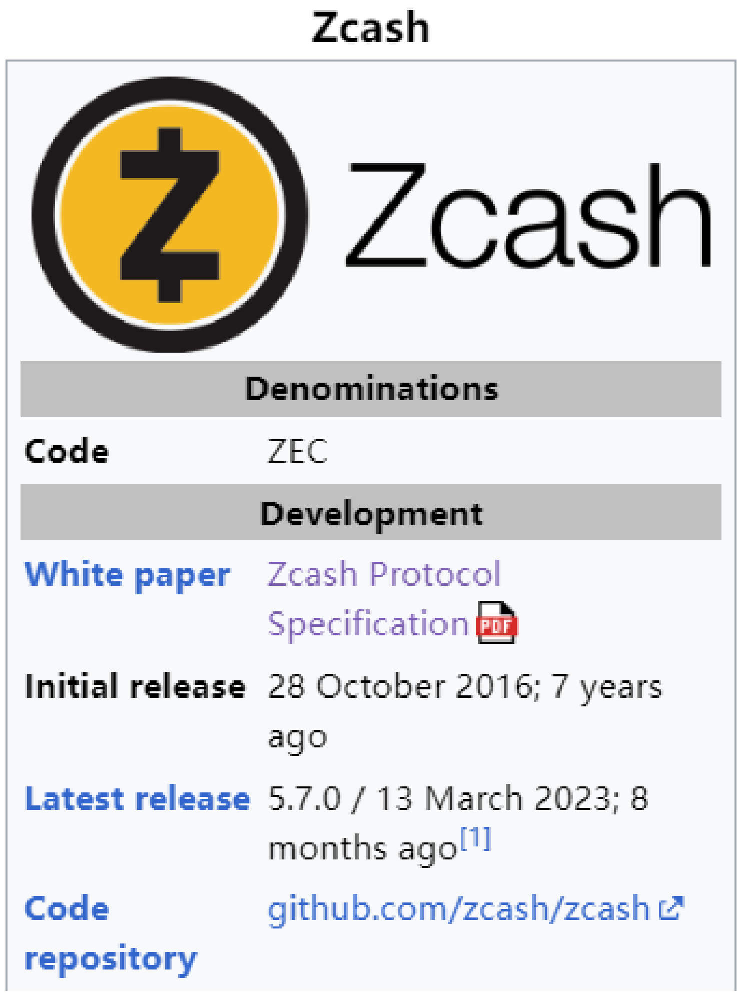
1) Parse $\mathsf{addr_{pk}}$ as $(a_{\mathsf{pk}}, \mathsf{pk_{enc}})$.
2) Randomly sample a $\mathsf{PRF}^{\mathrm{sn}}$ seed ρ.
3) Randomly sample two COMM trapdoors r, s.
4) Compute $k := \mathsf{COMM}_r(a_{\mathsf{pk}} \| \rho)$.
5) Compute $\mathsf{cm} := \mathsf{COMM}_s(v \| k)$.
6) Set $\mathbf{c} := (\mathsf{addr_{pk}}, v, \rho, r, s, \mathsf{cm})$.
7) Set $\mathsf{tx_{Mint}} := (\mathsf{cm}, v, *)$, where $* := (k, s)$.
8) Output $\mathbf{c}$ and $\mathsf{tx_{Mint}}$.

Summary

System	Type	Anonymity attacks	Deployability
Bitcoin	pseudonymous	transaction graph analysis	default
Manual mixing	mix	transaction graph analysis, bad mixes/peers	usable today
Chain of mixes or coinjoins	mix	side channels, bad mixes/peers	bitcoin-compatible
Zerocoin	cryptographic mix	side channels (possibly)	altcoin, trusted setup
Zerocash	untraceable	none known	altcoin, trusted setup

A comparison of the anonymity technologies

ZCash



Zcash

Denominations	
Code	ZEC
Development	
White paper	Zcash Protocol Specification
Initial release	28 October 2016; 7 years ago
Latest release	5.7.0 / 13 March 2023; 8 months ago[1]
Code repository	github.com/zcash/zcash

Zcash is an implementation of the Decentralized Anonymous Payment scheme Zerocash, with security fixes and improvements to performance and functionality. It bridges the existing transparent payment scheme used by Bitcoin with a shielded payment scheme secured by zero-knowledge succinctnon-interactive arguments of knowledge (zk-SNARKs). It attempted to address the problem of mining centralization by use of the Equihash memory-hard proof-of-work algorithm.

What is a zk-SNARK ?

SNARK: a succinct proof that a certain statement is true

Example statement: "I know an m such that SHA256$(m) = 0$"

- **SNARK**: the proof is **"short"** and **"fast"** to verify

  [if m is 1GB then the trivial proof (the message m) is neither]

- **zk-SNARK**: the proof "reveals nothing" about m

Blockchain Applications

Some applications require zero knowledge (privacy):

- **Private Tx on a public blockchain**:
  - zk proof that a private Tx is valid (Tornado cash, Zcash, IronFish, Aleo)

Blockchains drive the development of SNARKs

… but **<u>many</u>** non-blockchain applications benefit

Review: arithmetic circuits

Fix a finite field $\mathbb{F} = \{0, \ldots, p-1\}$ for some prime p>2.

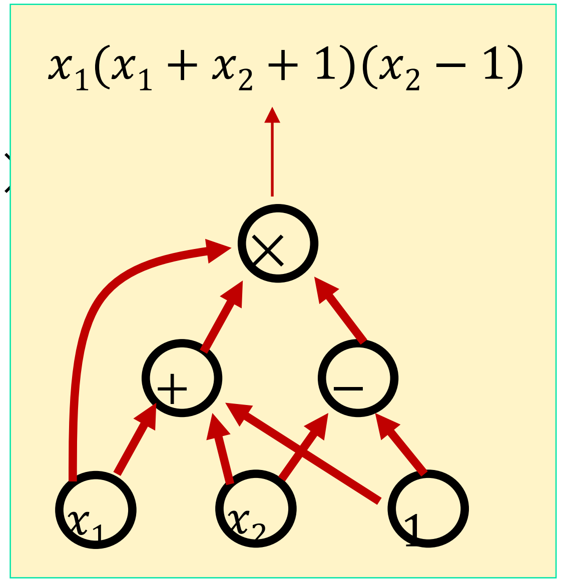
Arithmetic circuit: $C \colon \mathbb{F}^n \to \mathbb{F}$

- directed acyclic graph (DAG) where

  internal nodes are labeled $+, -,$ or

  inputs are labeled $1, x_1, \ldots, x_n$
- defines an n-variate polynomial with an evaluation recipe

$|C|$ = # gates in C



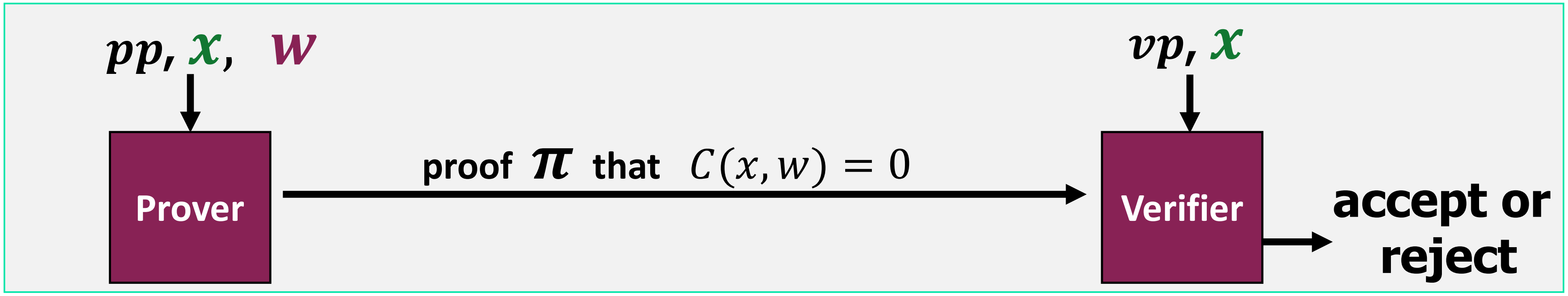
(preprocessing) NARK: Non-interactive ARgument of Knowledge

Public arithmetic circuit: $C(\boldsymbol{x}, \boldsymbol{w}) \rightarrow \mathbb{F}$

public statement in $\mathbb{F}^n$ $\rightarrow \boldsymbol{x}$

secret witness in $\mathbb{F}^m$ $\rightarrow \boldsymbol{w}$

Preprocessing (setup): S(C) → public parameters (pp, vp)



(preprocessing) NARK: Non-interactive ARgument of Knowledge

A **preprocessing NARK** is a triple (S, P, V):

- **S**(C) → public parameters (pp, vp) for prover and verifier
- **P**$(pp, \boldsymbol{x}, \boldsymbol{w})$ → proof π
- **V**$(vp, \boldsymbol{x}, \boldsymbol{\pi})$ → accept or reject

all algs. and adversary have access to a random oracle

NARK: requirements (informal)

Prover P(pp, $\boldsymbol{x}$, $\boldsymbol{w}$) **Verifier V (vp, $\boldsymbol{x}$, $\boldsymbol{\pi}$)**

$\xrightarrow{\textbf{proof } \pi}$ **accept or reject**

Complete: $\forall x, w$: $C(\boldsymbol{x}, \boldsymbol{w}) = 0 \Rightarrow$ **Pr[V(vp, x, P(pp, $\boldsymbol{x}$, $\boldsymbol{w}$)) = accept] = 1**

Adaptively knowledge sound: V accepts $\Rightarrow$ P "knows" $\boldsymbol{w}$ s.t. $C(\boldsymbol{x}, \boldsymbol{w}) = 0$

(an extractor E can extract a valid $\boldsymbol{w}$ from P)

Optional: Zero knowledge: $(C, pp, vp, \boldsymbol{x}, \pi)$ **"reveal nothing new" about $\boldsymbol{w}$**

(witness exists $\Rightarrow$ can simulate the proof)

SNARK: a <u>Succinct</u> AR**gument of Knowledge**

SNARK: **a NARC** **(complete and knowledge sound)** **that is <u>succinct</u>**

zk-SNARK: **a SNARK that is also zero knowledge**

Applications of SNARKs:

- Tornado cash: a zk-based mixer

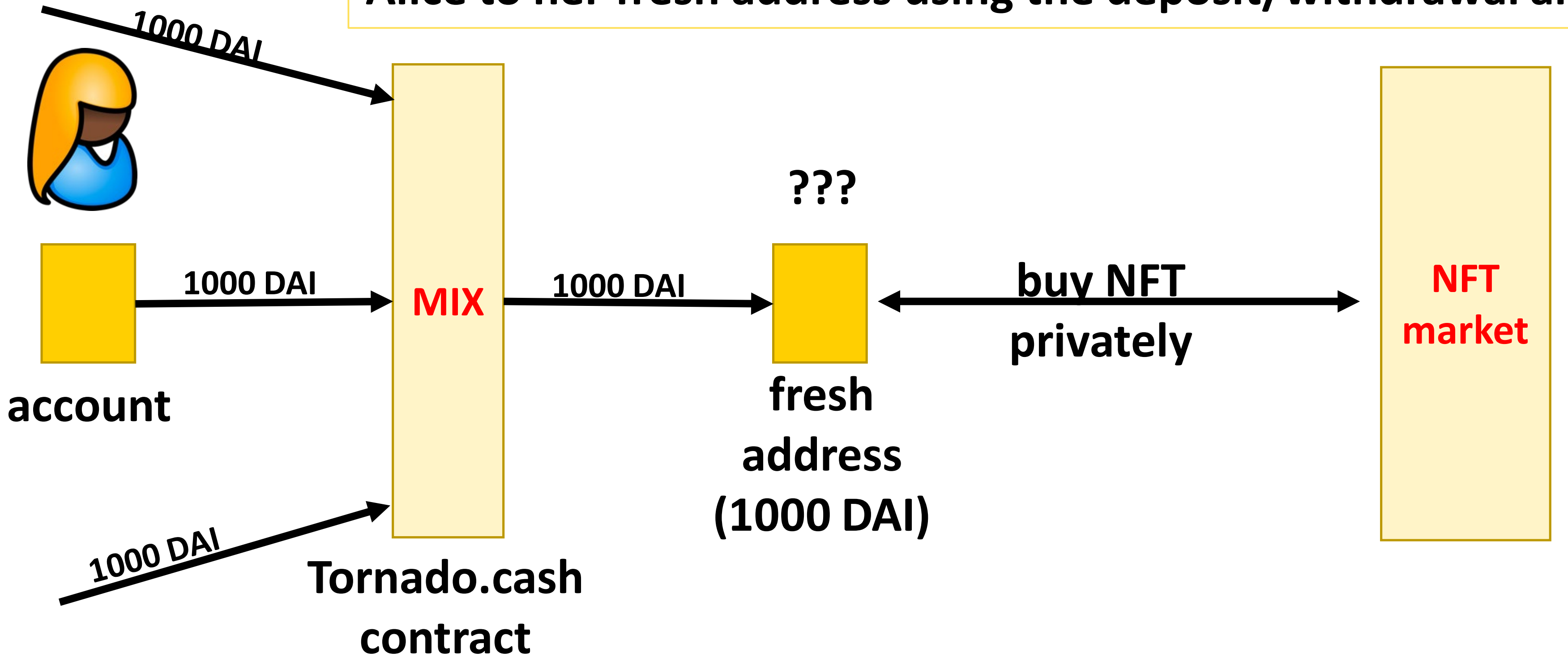
Launched on the Ethereum blockchain on May 2020 (v2)

Tornado Cash（代币为TORN）独辟蹊径，没有自建公链，Tornado是以太坊上的一个**隐私协议**，Tornado是基于零知识证明在以太坊上实现的隐私交易中间件。它也使用零知识证明技术，能够以不可追溯的方式将ETH以及ERC20代币发送到任何地址。

Tornado 隐私的是公链资产 VS zec & xmr

Tornado Cash: a ZK-mixer

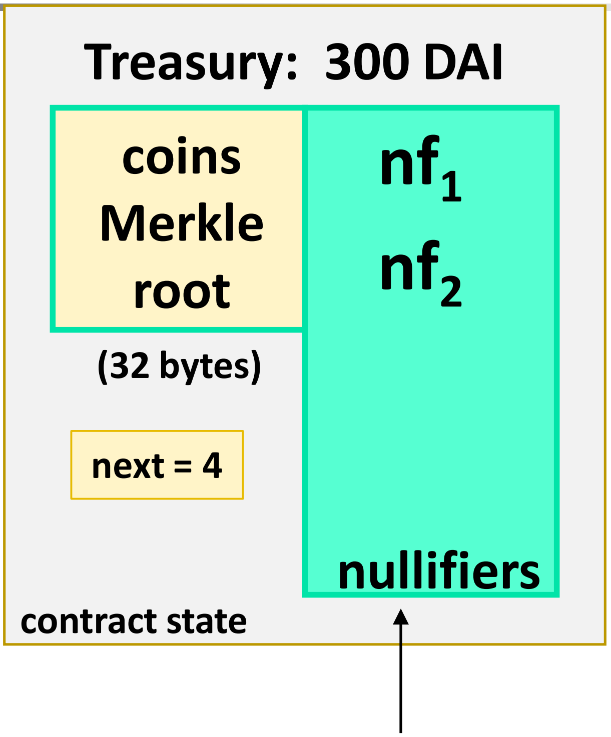
A common denomination (1000 DAI) is needed to prevent linking Alice to her fresh address using the deposit/withdrawal amount



The tornado cash contract(simplified)

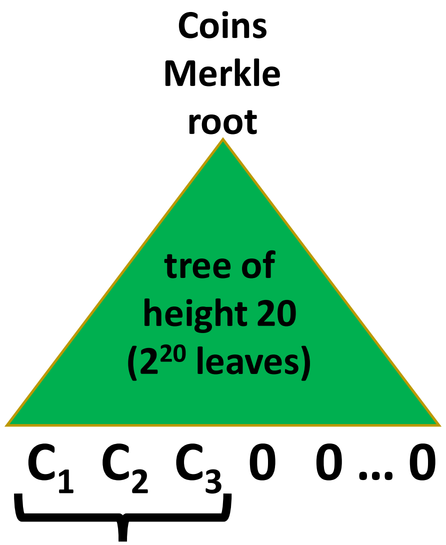
<u>100 DAI pool</u>:
each coin = 100 DAI

Currently:
- three coins in pool
- contract has 300 DAI
- two nullifiers stored



explicit list:
one entry per spent coin

$H_1, H_2\text{: } R \rightarrow \{0,1\}^{256}$ CRHF


Coins Merkle root
tree of height 20 (2^{20} leaves)
C_1 C_2 C_3 0 0 … 0

public list of coins

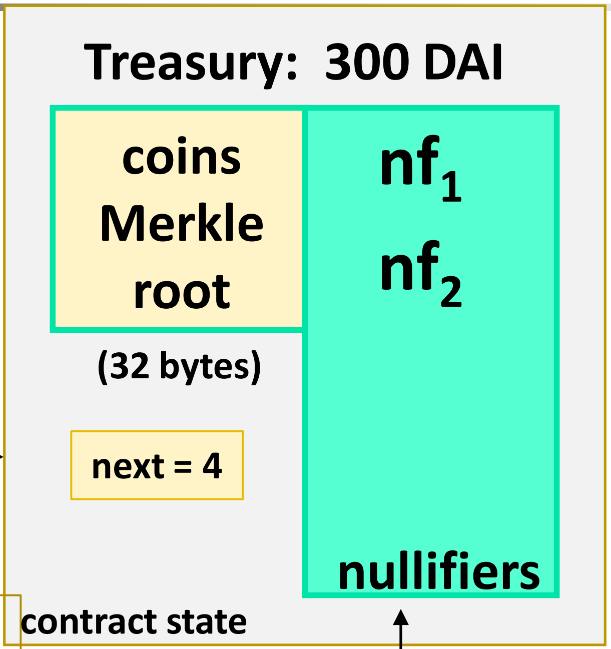
Tornado cash: deposit (simplified)

H_1, H_2: $R \to \{0,1\}^{256}$

100 DAI pool:
each coin = 100 DAI

Alice deposits 100 DAI:

100 DAI
C_4 , MerkleProof(4)

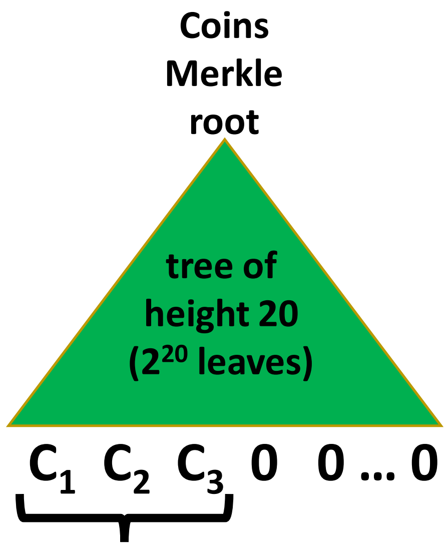


Build Merkle proof for leaf #4:
MerkleProof(4) (leaf=0)

choose random k, r in R

set $C_4 = H_1(k, r)$

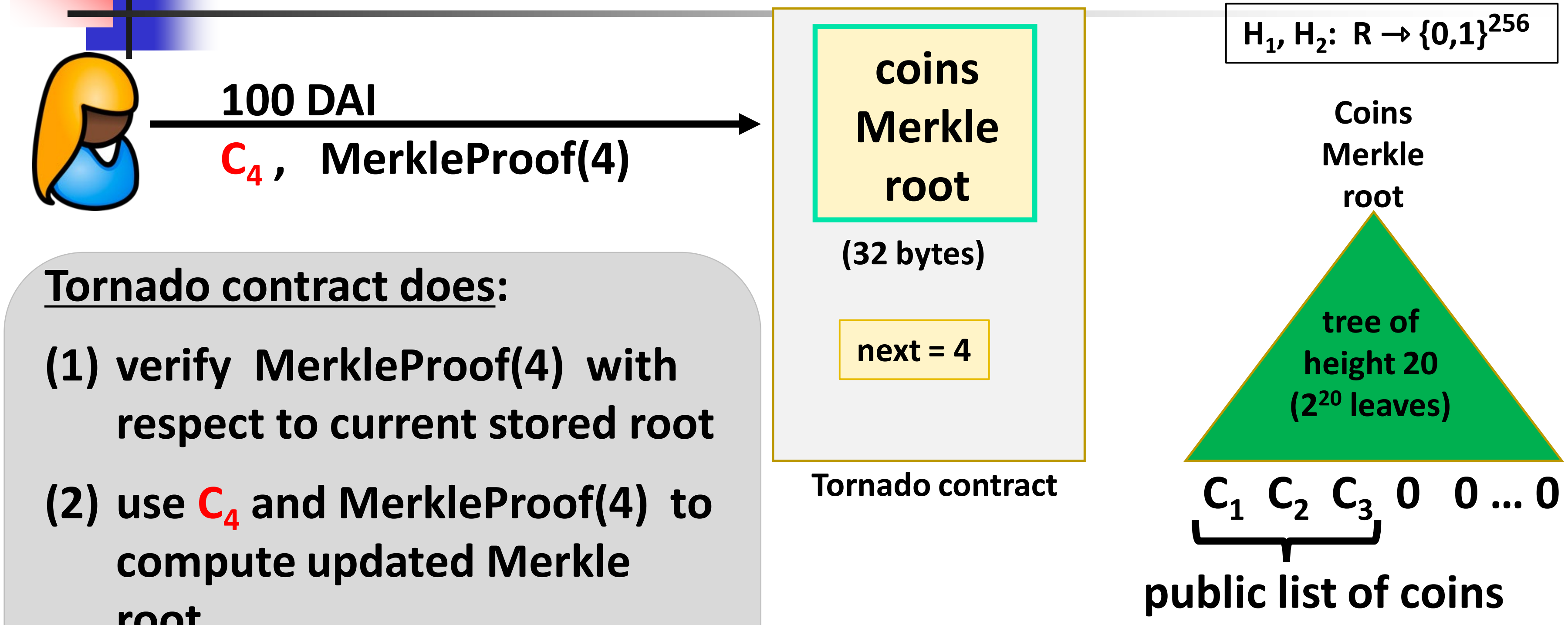
explicit list:
one entry per spent coin



Coins Merkle root
tree of height 20 (2^{20} leaves)
C_1 C_2 C_3 0 0 … 0

public list of coins

Tornado cash: deposit (simplified)



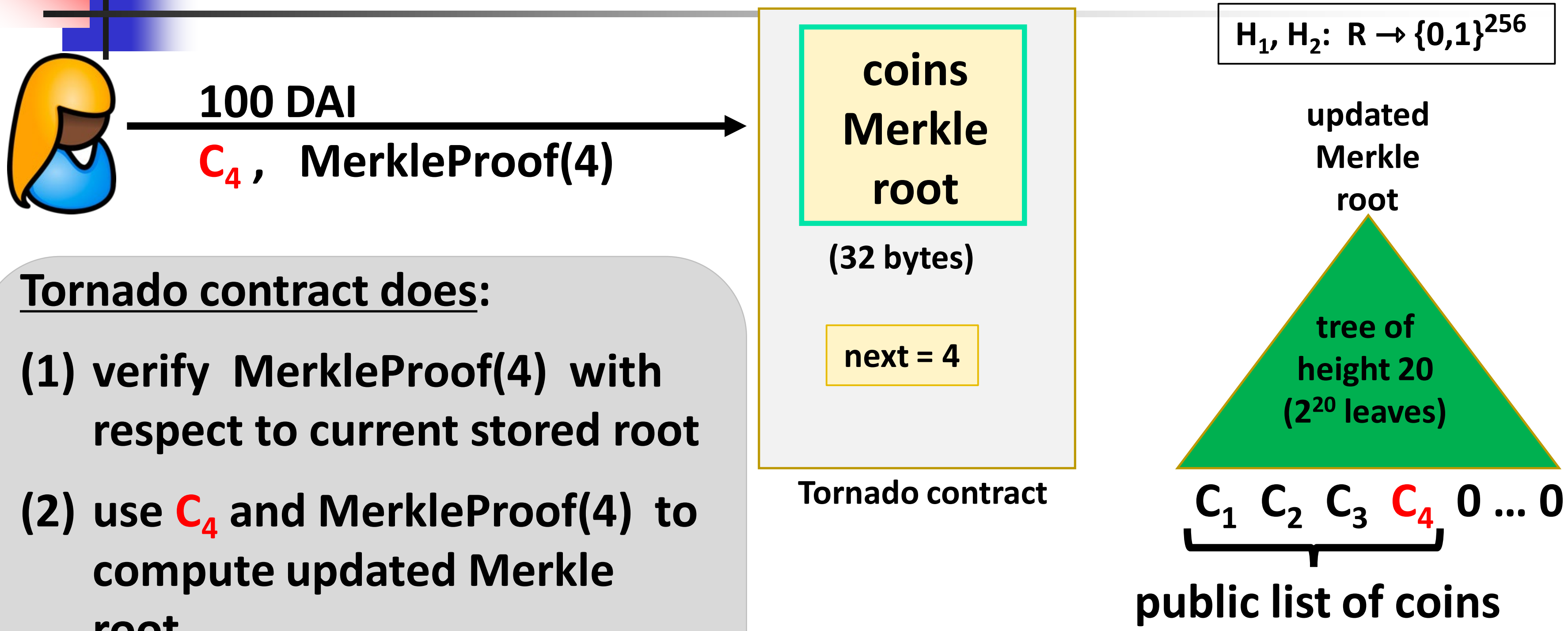
Tornado contract does:

(1) verify MerkleProof(4) with respect to current stored root

(2) use C_4 and MerkleProof(4) to compute updated Merkle root

(3) update state

Tornado cash: deposit (simplified)



Tornado contract does:

(1) verify MerkleProof(4) with respect to current stored root

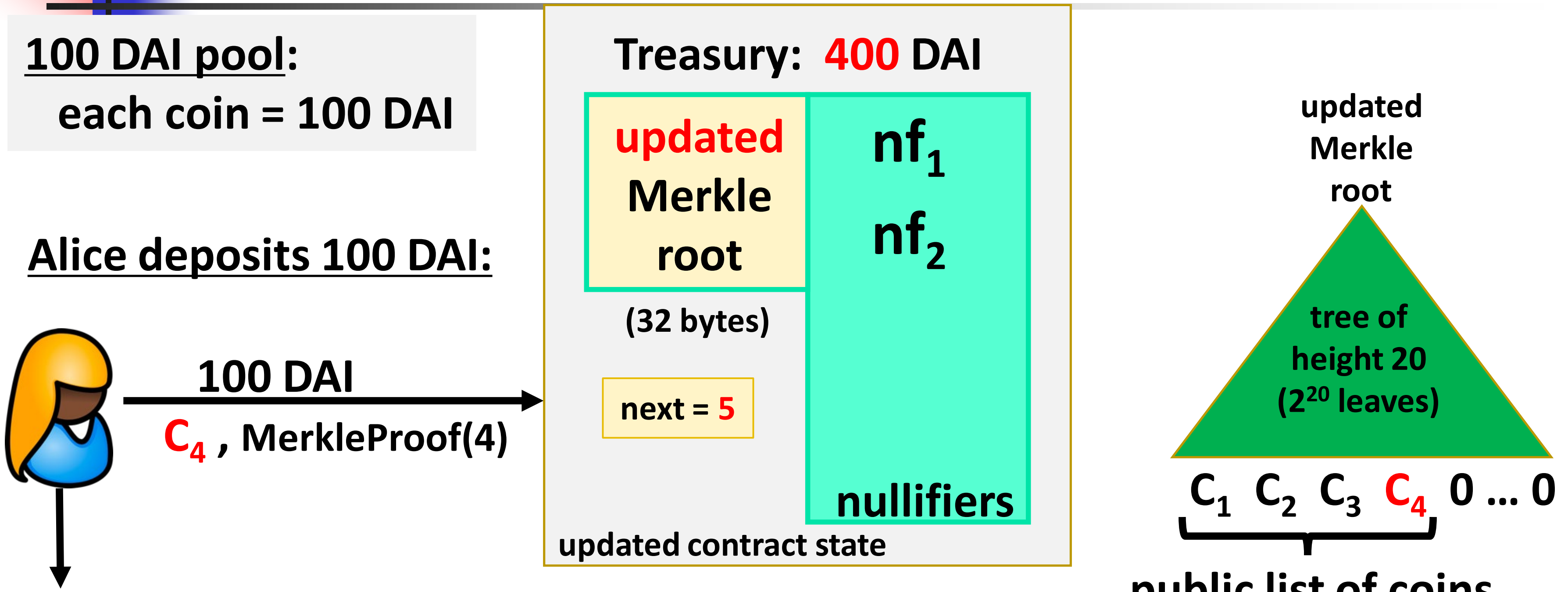
(2) use C_4 and MerkleProof(4) to compute updated Merkle root

(3) update state

Tornado cash: deposit (simplified)

100 DAI pool:
each coin = 100 DAI

Alice deposits 100 DAI:



public list of coins

note: (k, r)
Alice keeps secret
(one note per coin)

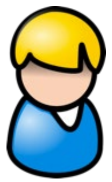
Every deposit: new Coin added sequentially to tree

an observer sees who owns which leaves

Tornado cash: withdrawal (simplified)

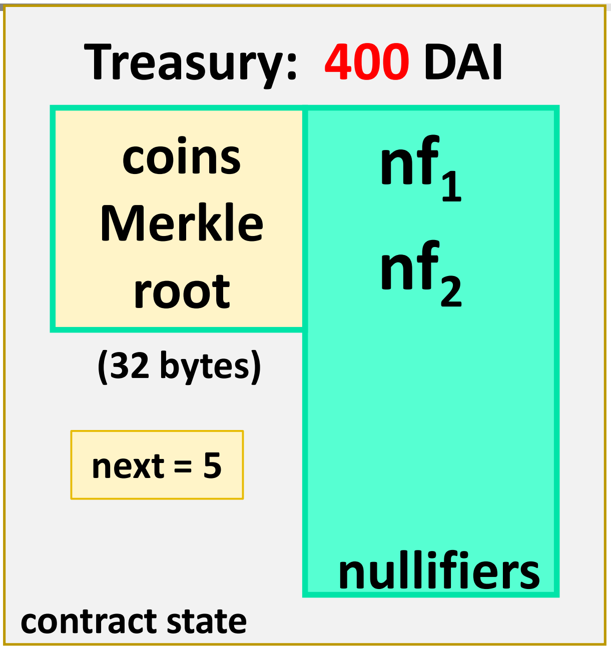
100 DAI pool:
each coin = 100 DAI

Withdraw coin #3 to addr A:

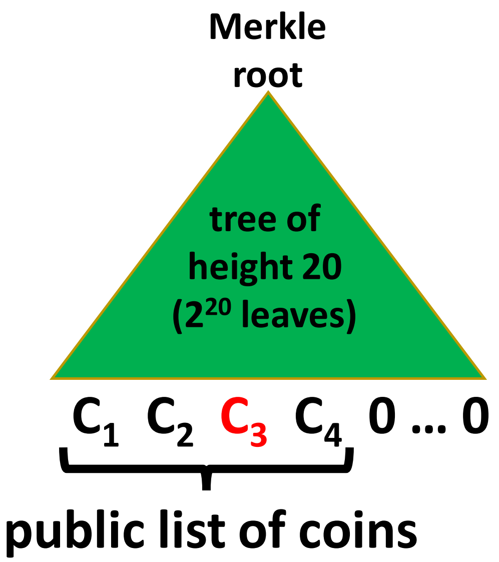


has note= (k′, r′)

set nf = $H_2(k')$



H_1, H_2: $R \to \{0,1\}^{256}$



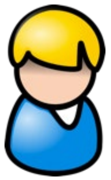
Merkle root
tree of height 20 (2^{20} leaves)
C_1 C_2 C_3 C_4 0 … 0
public list of coins

Bob proves "I have a note for some leaf in the coins tree, and its nullifier is nf" (without revealing which coin)

Tornado cash: withdrawal (simplified)

$H_1, H_2 : R \to \{0,1\}^{256}$

Withdraw coin #3 to addr A:



has note= (k', r')   set nf = H_2(k')

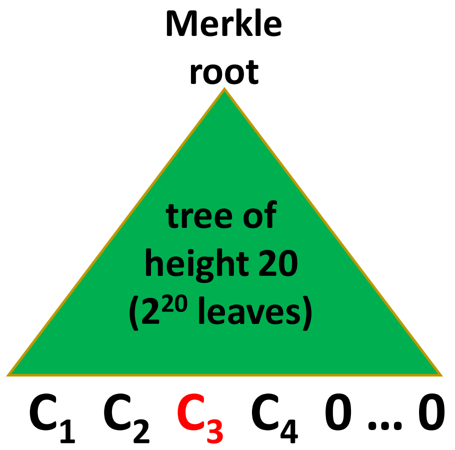
Bob builds zk-SNARK proof π for
public statement x = (root, nf, A)
secret witness w = (k', r', C_3, MerkleProof(C_3))

where Circuit(x,w)=0 iff:

(i) C_3 = (leaf #3 of root), i.e. MerkleProof(C_3) is valid,

(ii) $C_3 = H_1$(k', r'), and

(iii) nf = H_2(k').

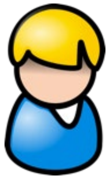

Merkle root
tree of height 20 (2^{20} leaves)
C_1 C_2 C_3 C_4 0 … 0

(address A not used in Circuit)

Tornado cash: withdrawal (simplified)

H_1, H_2: $R \rightarrow \{0,1\}^{256}$

Withd



The address A is part of the statement to ensure that a miner cannot change A to its own address and steal funds

Assumes the SNARK is non-malleable:

adversary cannot use proof π for x to build a proof π' for some "related" x'

(e.g., where in x' the address A is replaced by some A')

C_1 C_2 C_3 C_4 0 … 0

Bob builds zk-SNARK proof π for

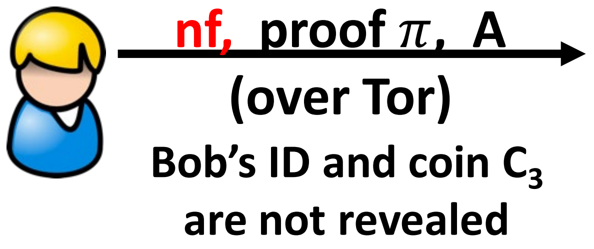
public statement x = (root, nf, A)

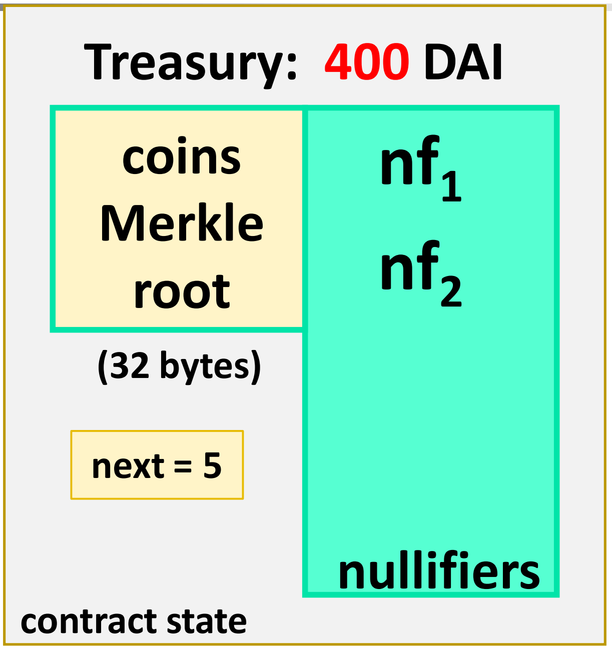
secret witness w = (k', r', C_3, MerkleProof(C_3))

Tornado cash: withdrawal (simplified)

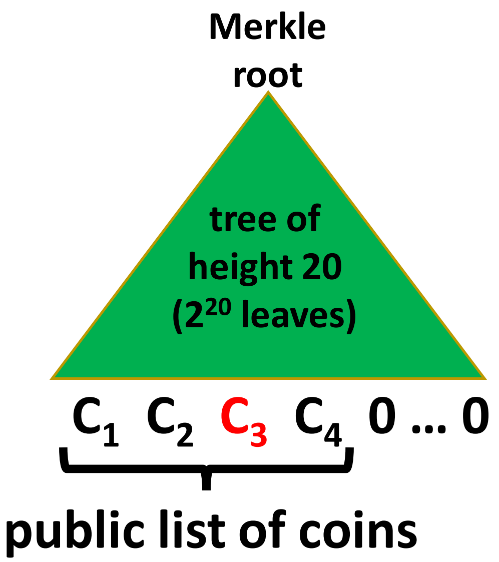
100 DAI pool:
each coin = 100 DAI

Withdraw coin #3 to addr A:





H_1, H_2: $R \to \{0,1\}^{256}$

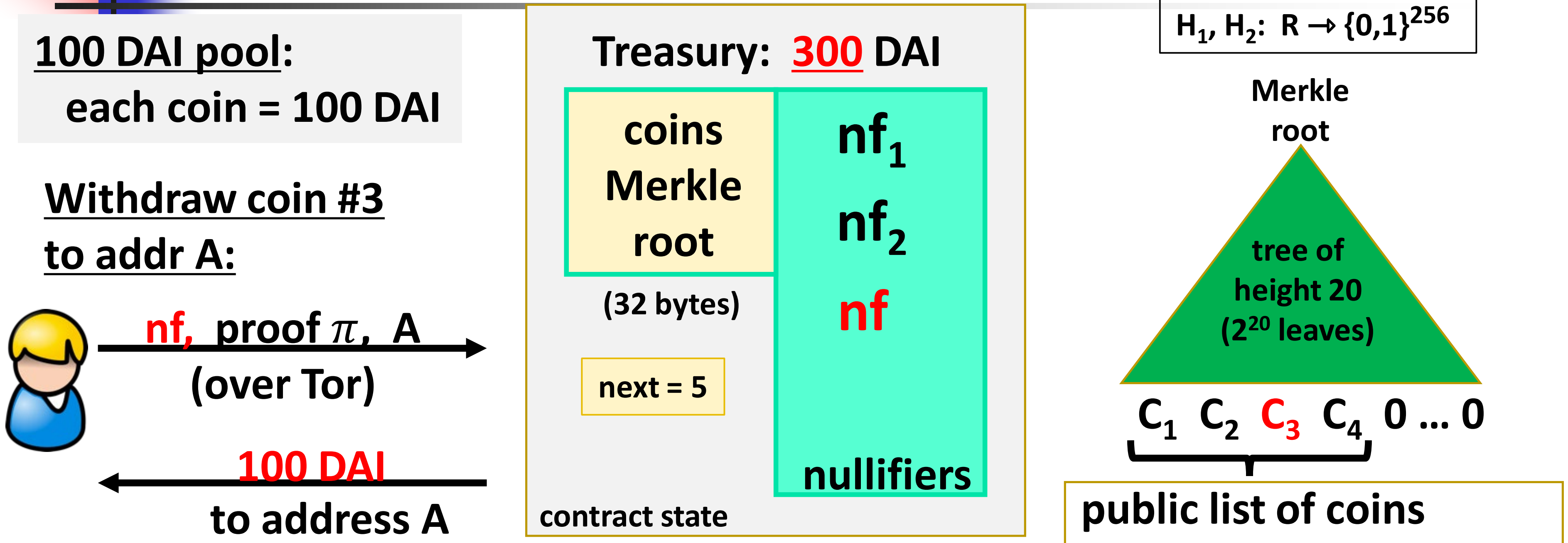

Merkle root
tree of height 20 (2^{20} leaves)
C_1 C_2 C_3 C_4 0 … 0
public list of coins

Contract checks (i) proof π is valid for (root, nf, A), and
(ii) nf is not in the list of nullifiers

Tornado cash: withdrawal(simplified)

100 DAI pool:
each coin = 100 DAI

Withdraw coin #3 to addr A:



public list of coins
… but observer does not know which are spent

nf and π reveal nothing about which coin was spent.

But, coin #3 cannot be spent again, because $nf = H_2(k')$ is now nullified.