

ZK SHANGHAI
零知识证明工作坊

递归和组合； 应用ZK结构 1

现代零知识密码学

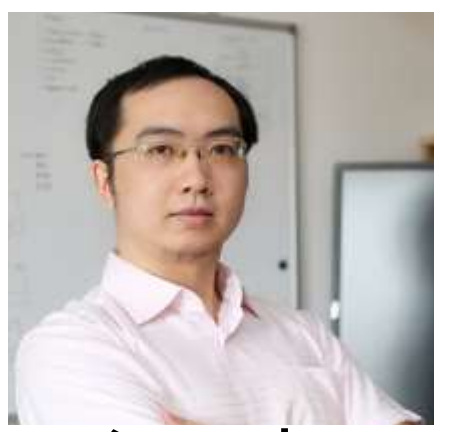
Hosted by [SutuLabs](#) & [Kepler42B-ZK Planet](#)

课程资源: zkshanghai.xyz

WORKSHOP!



个人介绍



梁爽

区块链 架构师

上海交大 计算机博士生
(休学创业中)

微信: icerdesign
微博: @wizicer
Github: @wizicer
Twitter: @icerdesign
LinkedIn: www.linkedin.com/in/icerdesign

- 1999年**
 - 正式开始学习写程序
- 2009年**
 - 在新媒传信（飞信）做高性能服务器程序架构及开发
- 2012年**
 - 在Honeywell工业控制部门做PLC、RTU上位机组态软件架构及开发
- 2017年**
 - 接触区块链，并开始创业开发区块链数据库
- 2020年**
 - 入学上海交大攻读博士学位，研究零知识证明数据库
- 2022年**
 - 获Chia全球开发大赛第一名，并开始Pawket钱包的开发
- 2023年**
 - 获得零知识链Mina的项目资助

今日课程内容

- 递归和组合
- 应用ZK结构 1
 - 成员资格证明

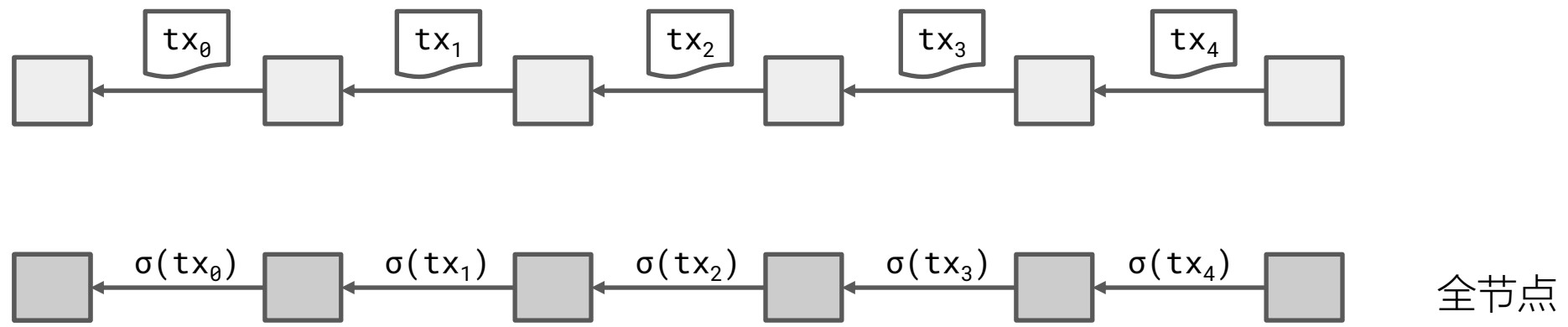
今日课程将回答以下问题

- 如何提高zkEVM的效率
- Tornado Cash原理
- Nullifier的用途
- 如何利用邮件证明资产

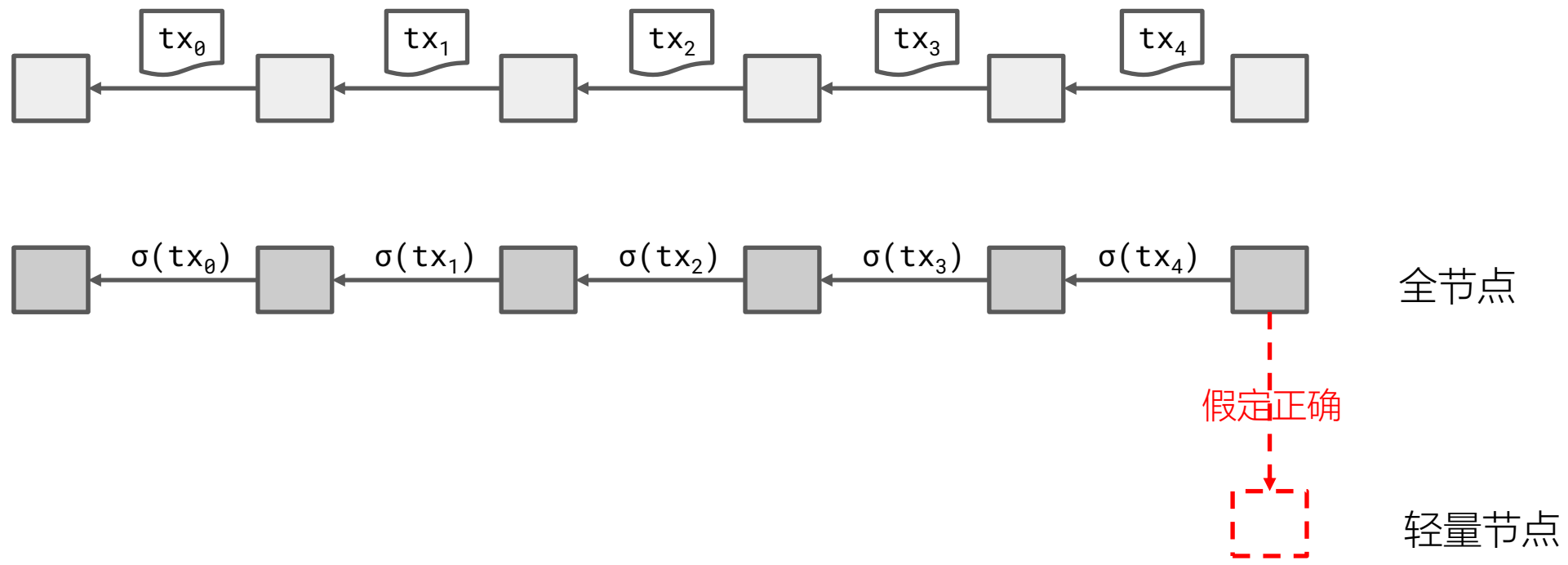
递归和组合

- 递归使得复杂计算的证明可以并行化。
- 证明组合将来自不同证明系统的子协议嵌在一起。
- 递归+证明组合= 🔥

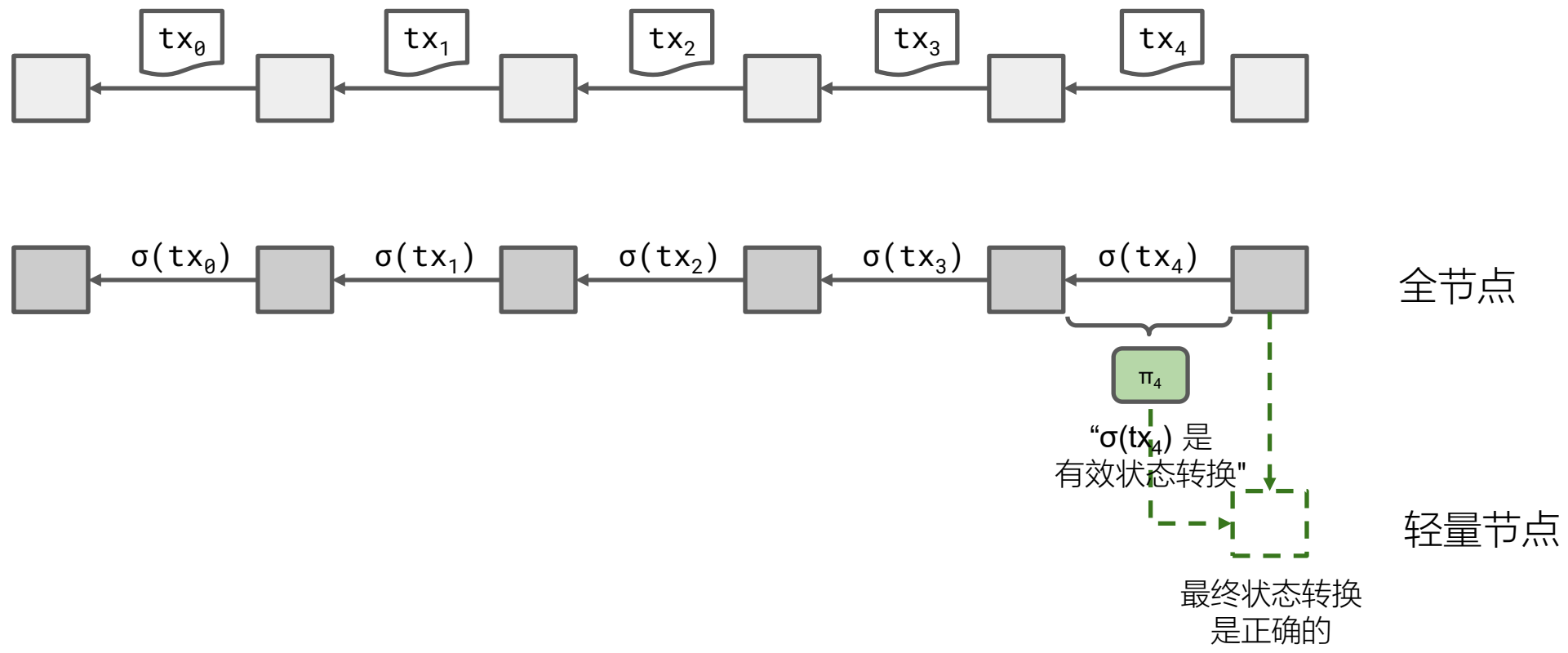
应用：每个区块可在常数时间内验证的区块链



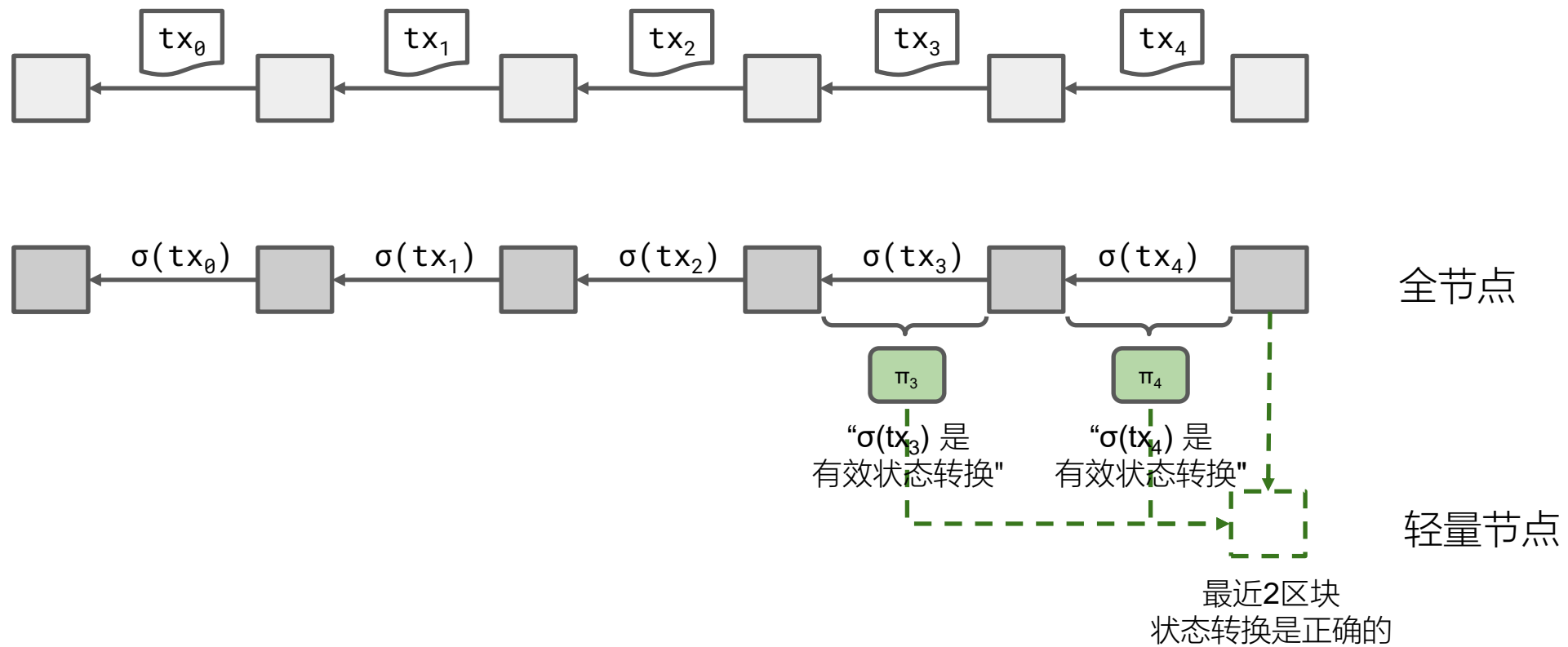
应用：每个区块可在常数时间内验证的区块链



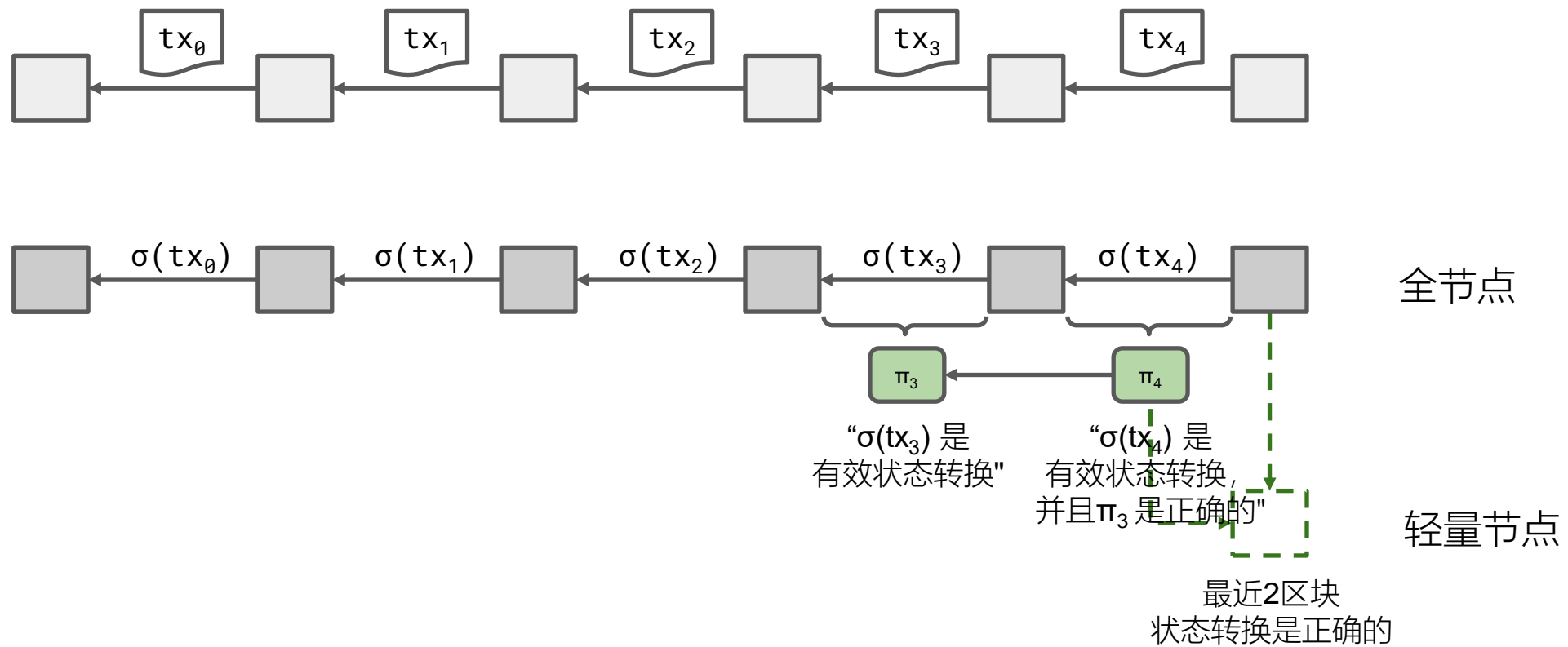
应用：每个区块可在常数时间内验证的区块链



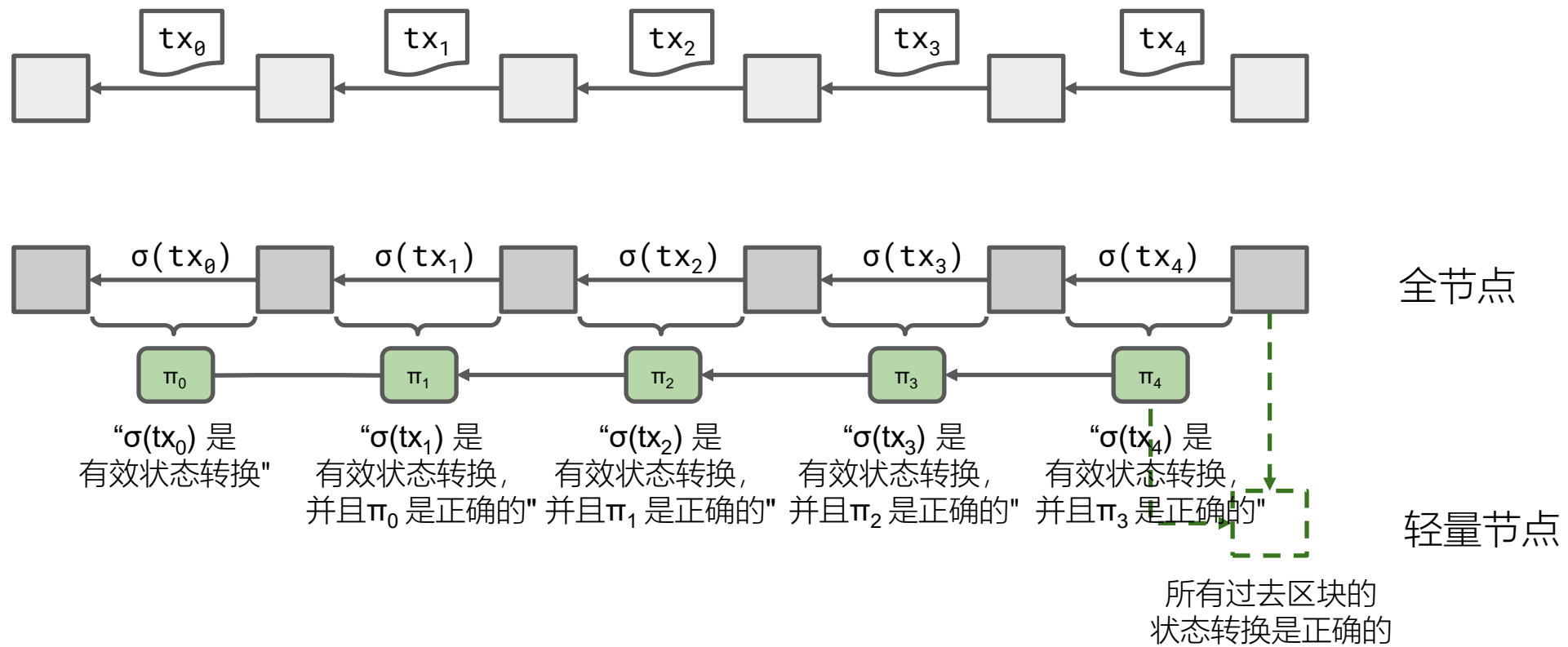
应用：每个区块可在常数时间内验证的区块链



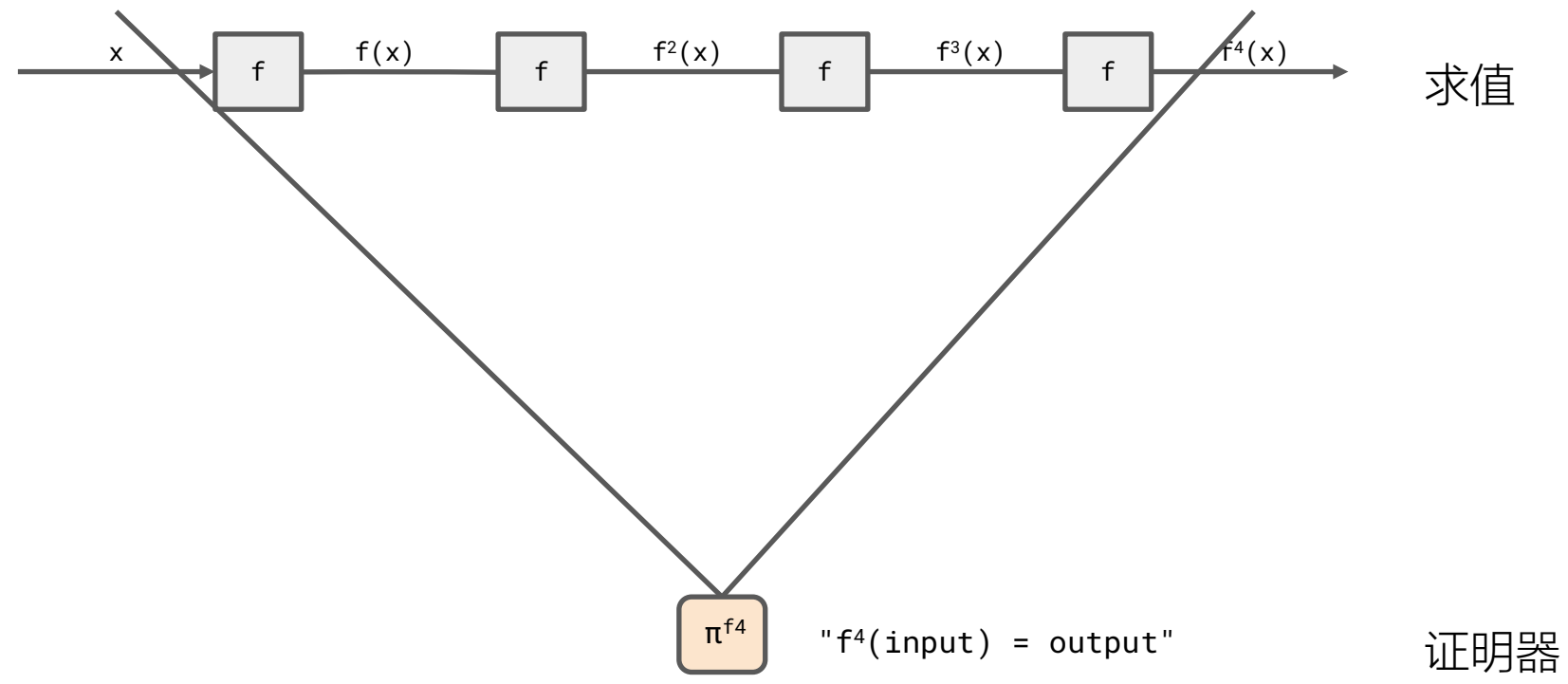
应用：每个区块可在常数时间内验证的区块链



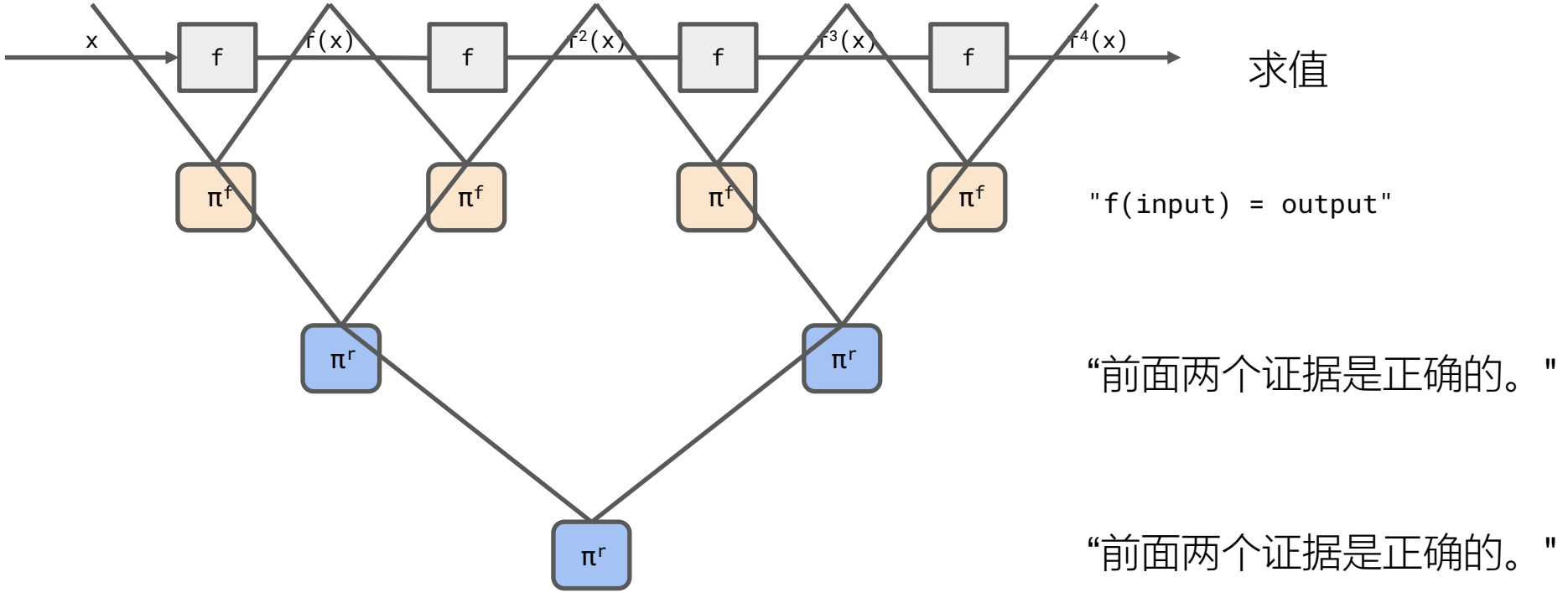
应用：每个区块可在常数时间内验证的区块链



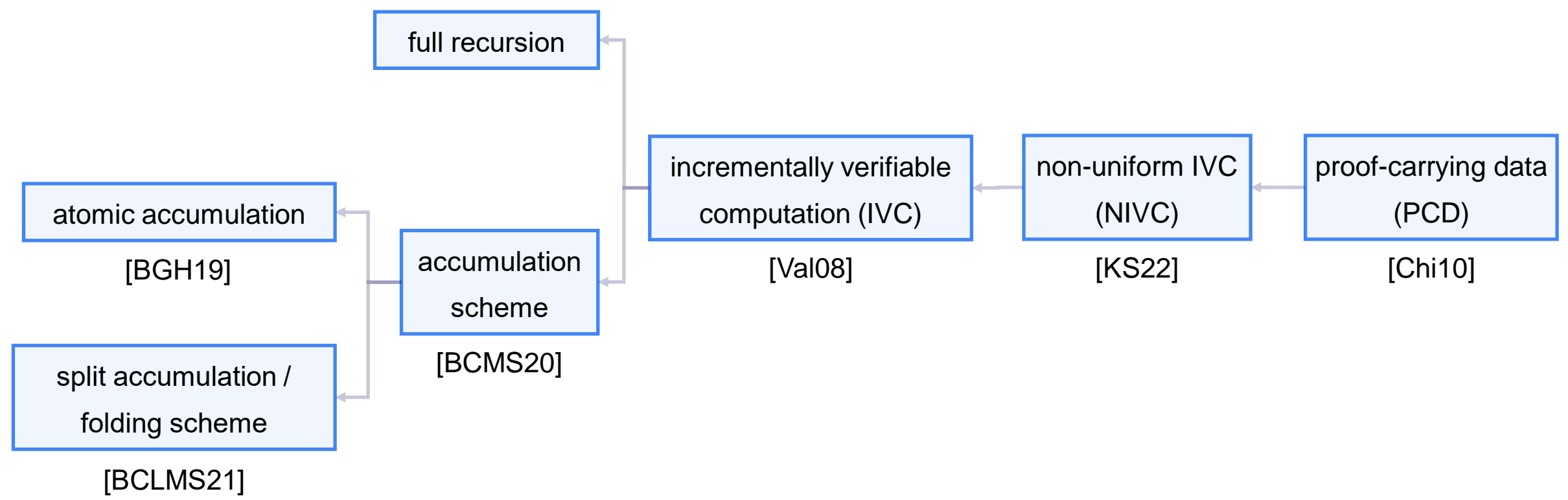
可验证延迟函数[BBBF18]



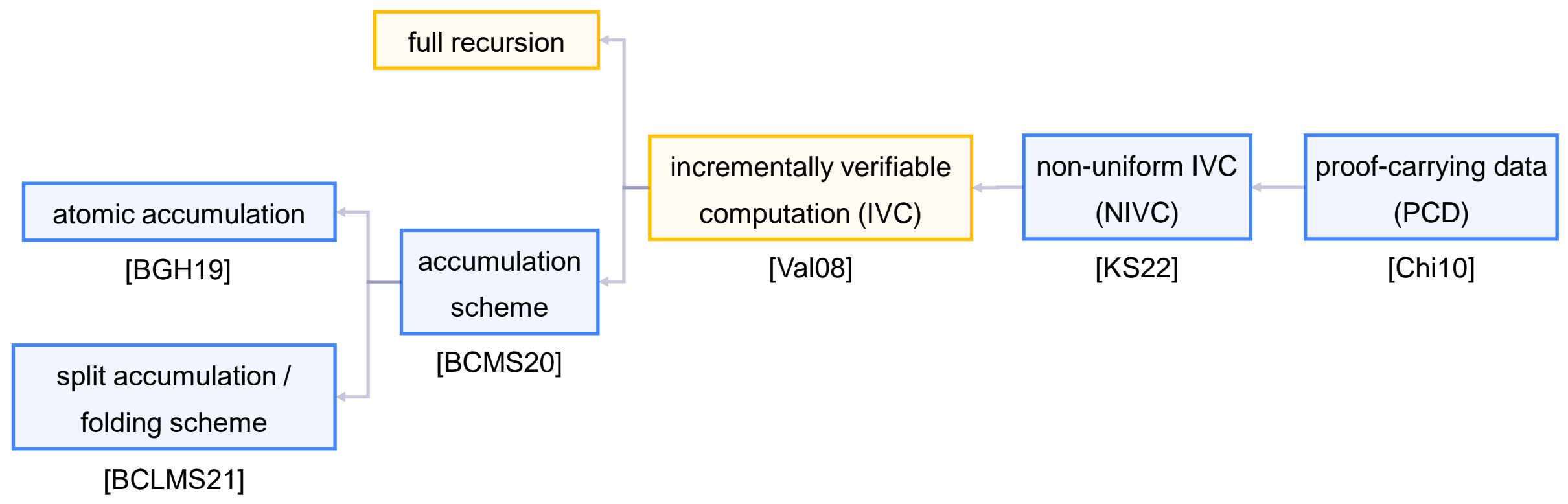
可验证延迟函数[BBBF18]



递归技术

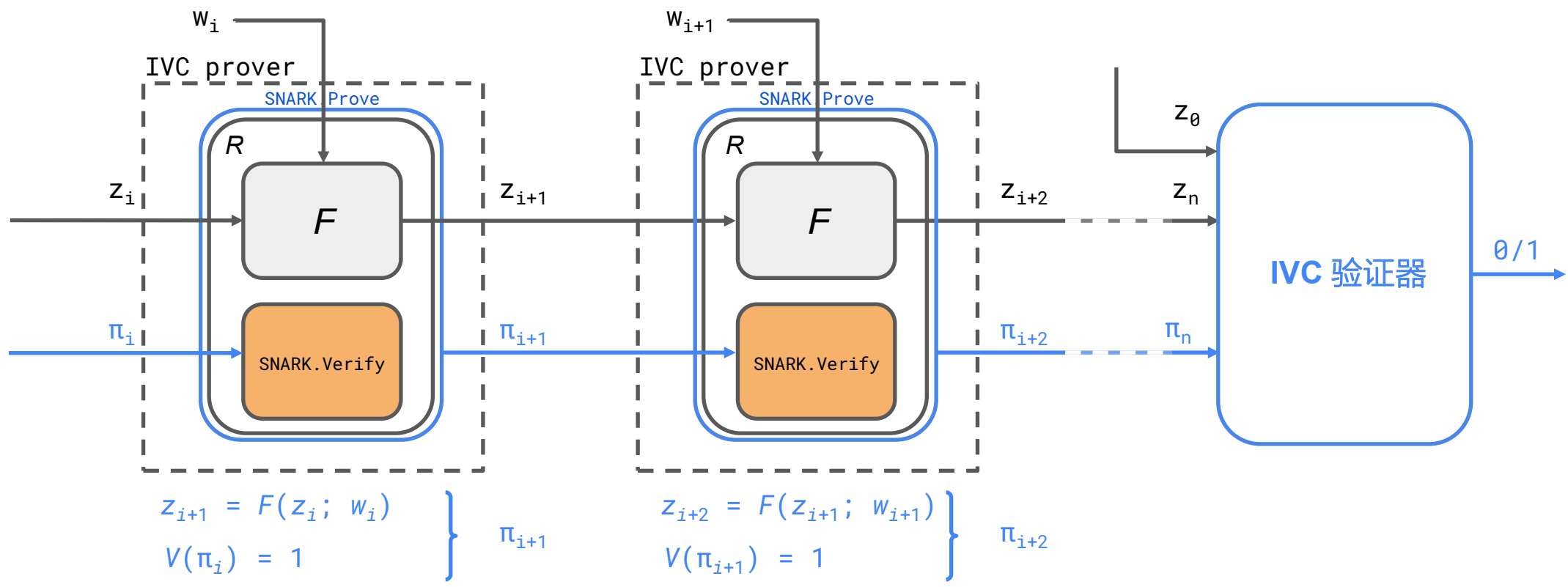


递归技术



递归技术： IVC完全递归

策略：将 $z_n = F^{(n)}(z_0; w_0, \dots, w_{n-1})$ 分解为 F 的递归应用。



递归技术：IVC完全递归

范例: plonky2

```
Initial proof degree 16384 = 2^14
Degree before blinding & padding: 4028
Degree after blinding & padding: 4096
```

```
// Start with a dummy proof of specified size
```

```
let inner = dummy_proof::<F, C, D>(config, log2_inner_size)?;
let (_, _, cd) = &inner;
```

```
Single recursion proof degree 4096 = 2^12
Degree before blinding & padding: 3849
Degree after blinding & padding: 4096
```

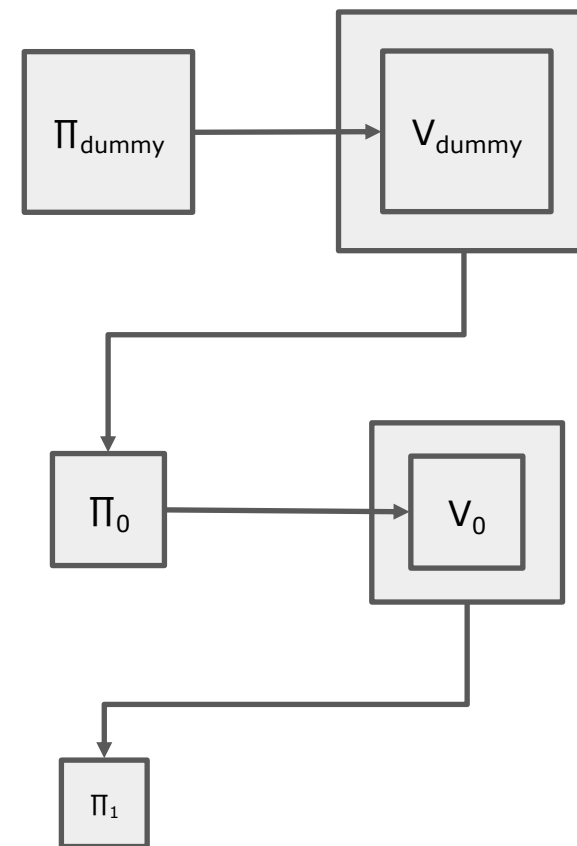
```
// Recursively verify the proof
```

```
let middle = recursive_proof::<F, C, C, D>(&inner, config, None)?;
let (_, _, cd) = &middle;
```

```
Double recursion proof degree 4096 = 2^12
Proof length: 127184 bytes
0.2511s to compress proof
Compressed proof length: 115708 bytes
```

```
// Add a second layer of recursion to shrink the proof size further
```

```
let outer = recursive_proof::<F, C, C, D>(&middle, config, None)?;
let (proof, vd, cd) = &outer;
```



组合技术：递归验证

	快速证明器	小证据 / 快速验证器
STARK	✓	✗
Groth16	✗	✓



组合技术：递归验证

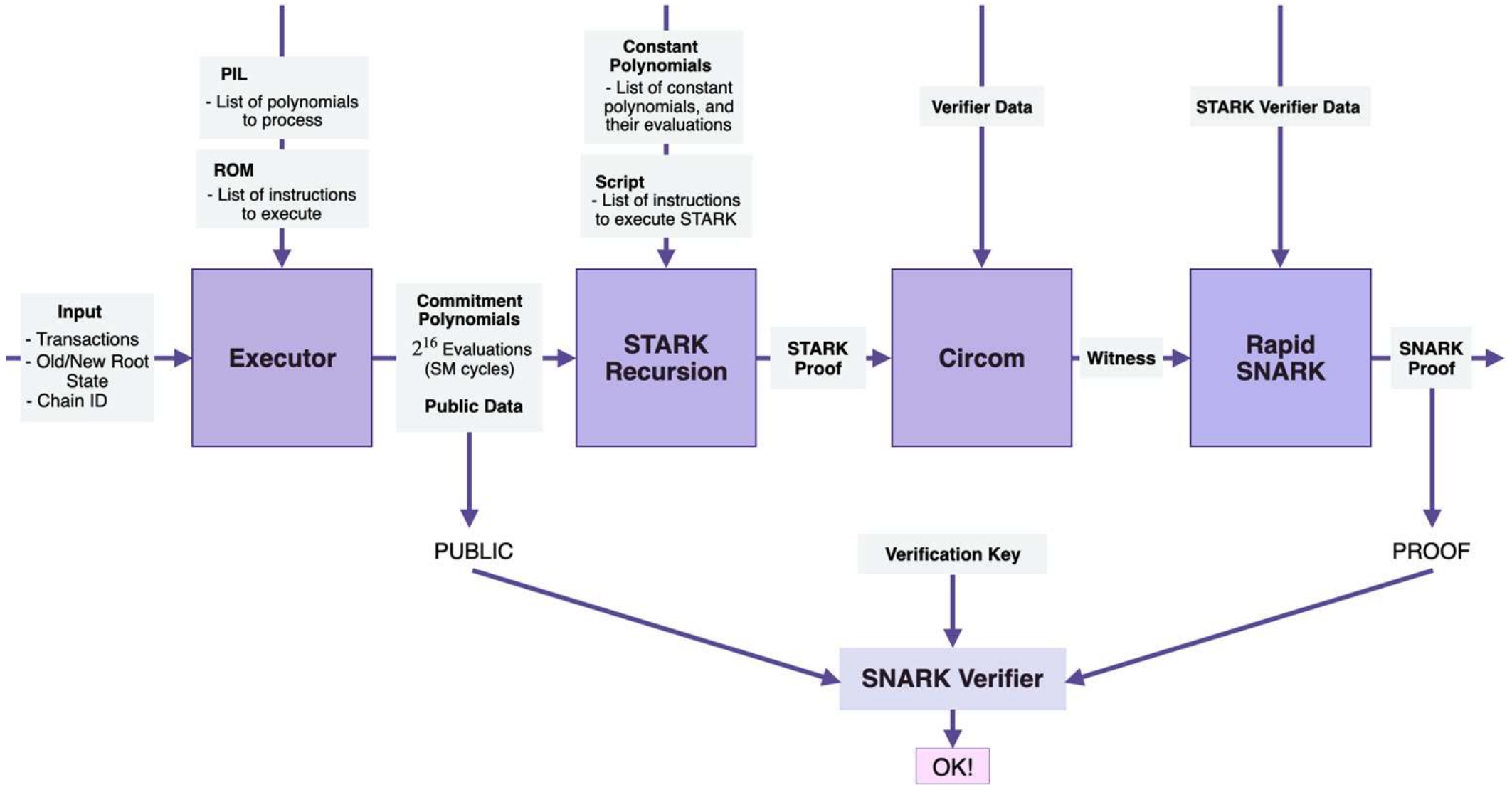
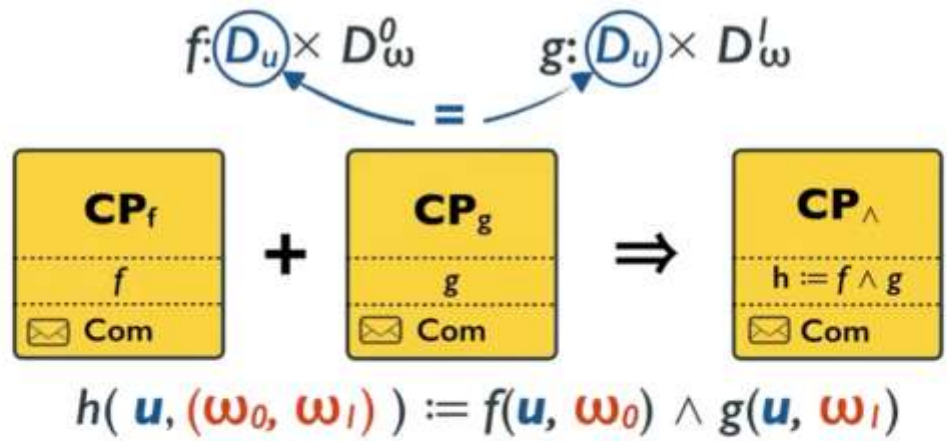


diagram from <https://docs.hermez.io/zkEVM/zkProver/State-Machines/Overview/zkProver-State-Machines/#the-zk-snark-prover>

组合技术：可链接的提交并证明

example: LegoSNARK [CFQ19]



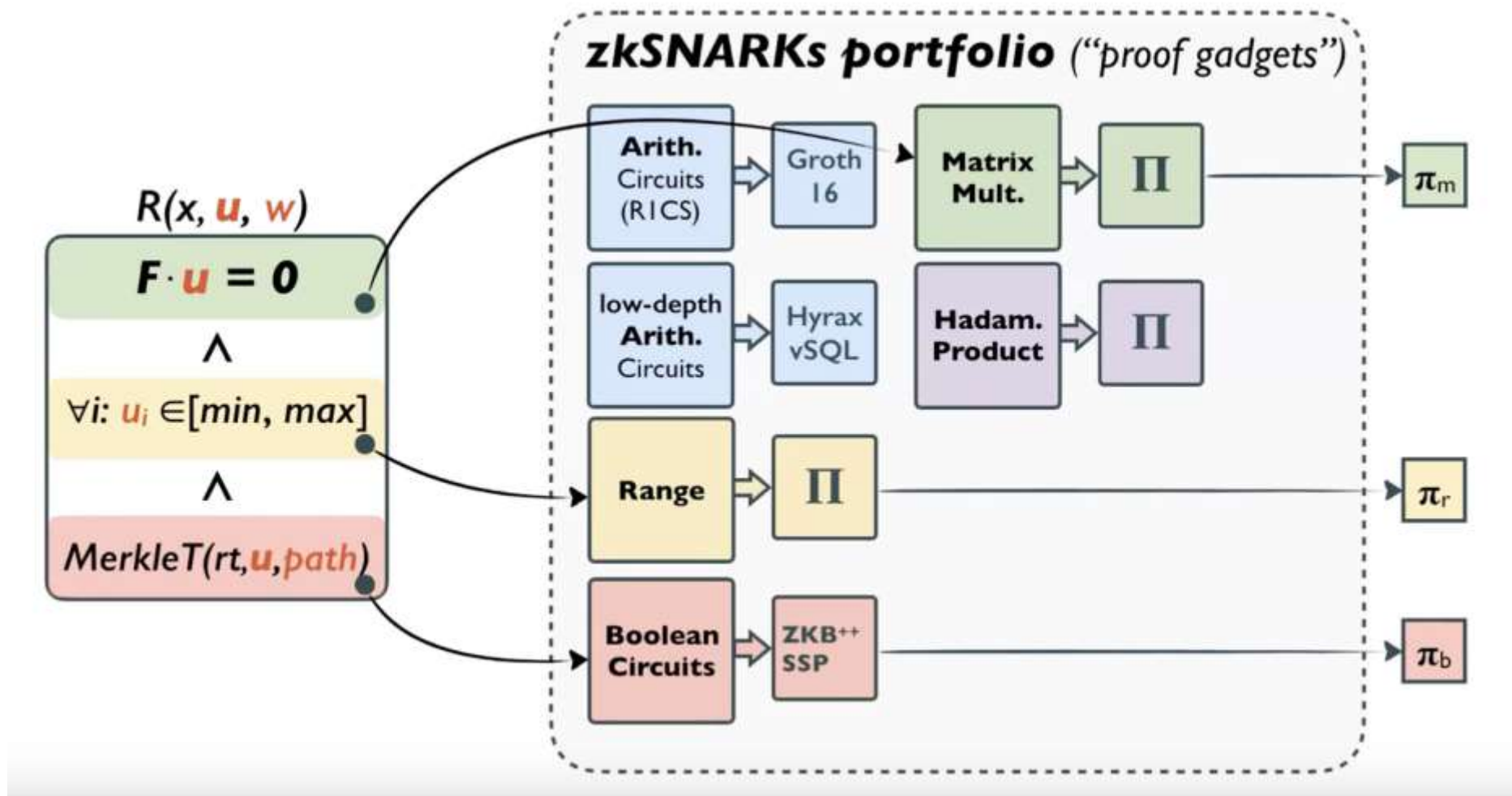
simple idea. $\pi_{\wedge} = (\text{Com}_{\text{ck}}(\mathbf{u}), \pi_f, \pi_g), \pi_f \leftarrow \text{CP}_f, \pi_g \leftarrow \text{CP}_g$

other compositions. disjunction, sequential composition, >2 relations

main message. focus on constructing proof gadgets, security is proven once for all

组合技术：可链接的提交并证明

example: LegoSNARK [CFQ19]



未来方向：项目想法

- 基准测试
 - 多项式承诺基准测试 ([2π.com/23/pc-bench/](https://2pi.com/23/pc-bench/))
 - ZKP编译器比赛 (github.com/anoma/zkp-compiler-shootout)
 - 什么是理想的“内部”证明？和理想的“外部”证明
- 库、标准、开发工具
 - LegoSNARK: github.com/imdea-software/legosnark
 - arkworks: [arkworks-rs/accumulation](https://arkworks-rs.github.io/accumulation), [arkworks-rs/pcd](https://arkworks-rs.github.io/pcd)
 - 我们能否为递归/组合策略定义高级API？
- 安全分析
 - 打破递归plonky2的正确性的最小示例？
 - 递归组合的安全性：需要更强的知识提取器（必须在每个递归步骤中都成功）
 - 利用统一框架简化分析

实用ZK构造

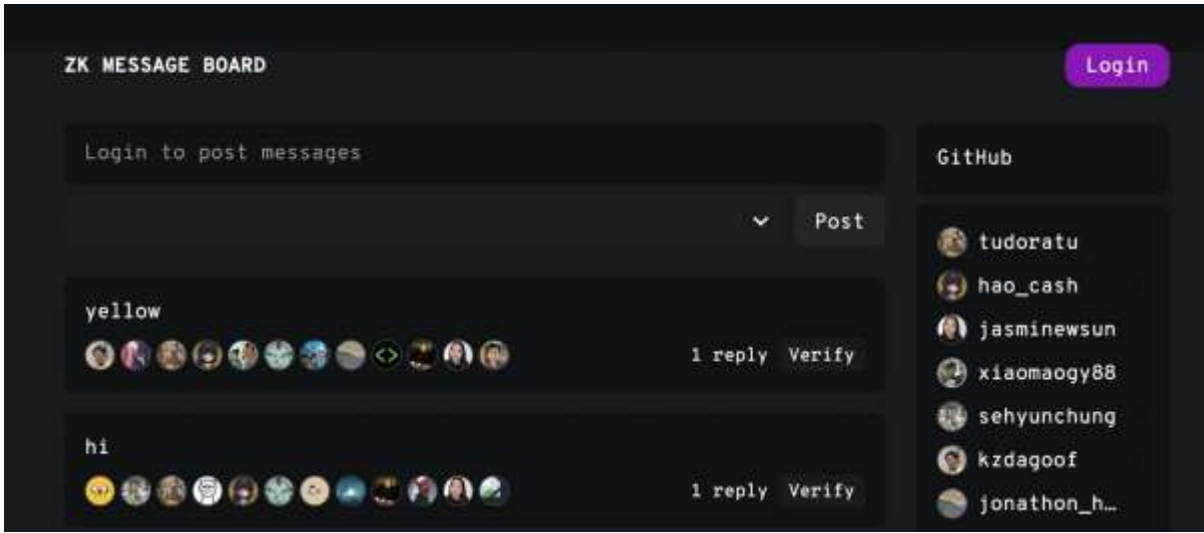
群成员资格 Group Membership

群成员资格证明 实现流程

- Group Membership (无 nullifiers)
 - zkmessage
 - heyanon
- Group Membership (有 nullifiers)
 - semaphore
 - tornado cash
- Group Membership (以太坊上的范例, 有 nullifiers)
 - stealthdrop
 - ethereum nullifiers
- Group Membership (其他范例)
 - zk email
 - ~~zk jwt~~

zkmessage.xyz

- 0xparc.org/blog/zk-group-sigs
 - kevin z, uma r, raymond z, joel g
- 通过每条消息附带的 zk 成员证明，代表一个群组发布消息。



spec: zkmessage.xyz



Database

joe1 -> 75...gk
uma -> 6h...l4

注册阶段



joel 生成 $\text{hash}(9j\dots46)=75\dots\text{gk}$
其中的秘密字符串: 9j...46



uma 生成 $\text{hash}(36\dots\text{f9})=6h\dots\text{l4}$
其中的秘密字符串: 36...f9

spec: zkmessage.xyz

发消息阶段



joel 生成ZK证明:

```

component prove:
  public input hashes[n]
  public input message

  private input secret

  matches = 0
  for i in 0...n:
    if hash(secret) == hashes[i]:
      matches++
  matches == 1

  prove(9j...46, [75...gk, 6h...14], "hi")

```



Database

joe1 -> 75...gk
uma -> 6h...14

spec: zkmessage.xyz

发消息阶段



Database

joel 生成ZK证明:

joe1 -> 75...gk
uma -> 6h...14

```

component prove:
  public input hashes[n]
  public input message

  private input secret

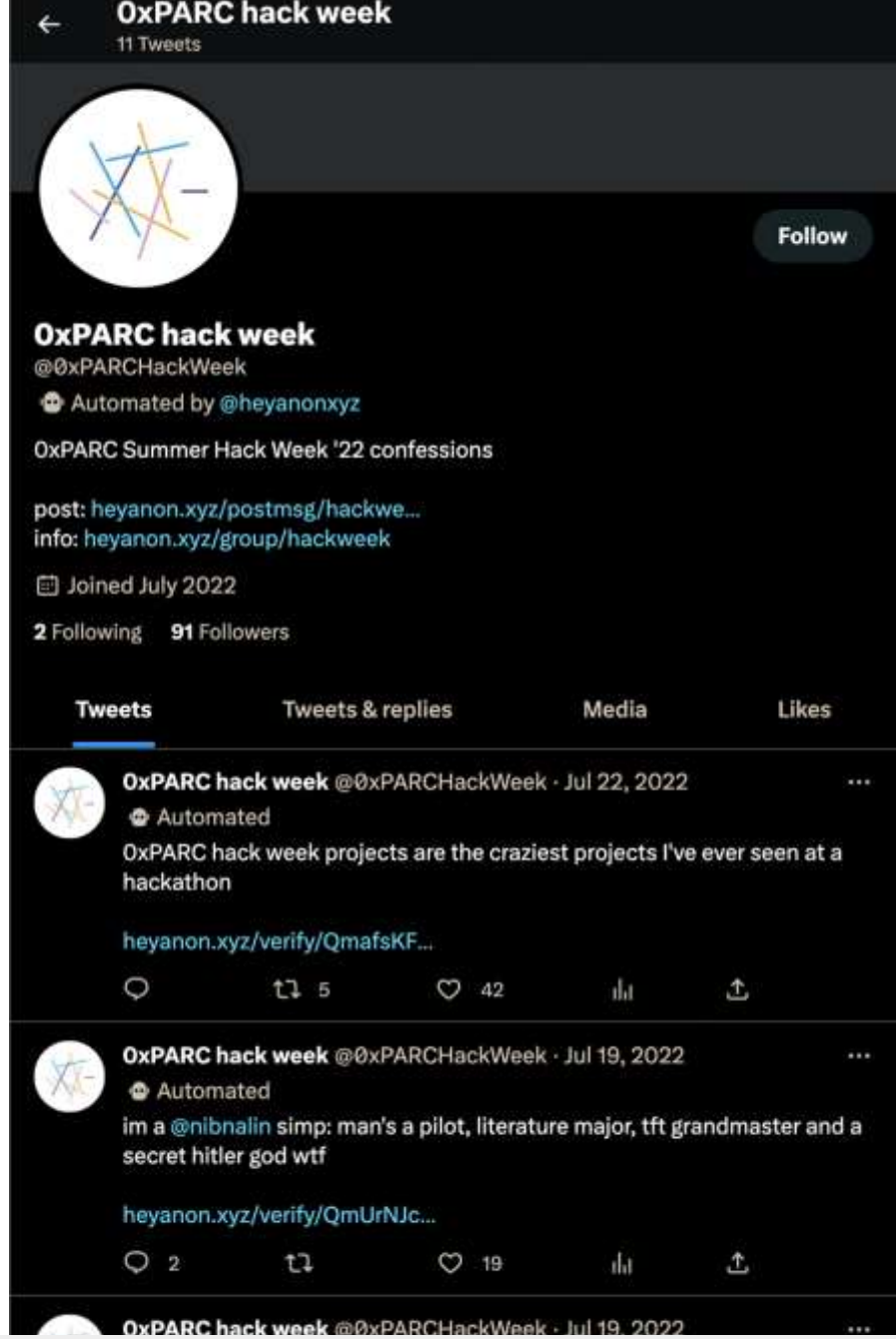
  (hash(secret) - hashes[0]) * (hash(secret) - hashes[1]) * ... == 0

prove(9j...46, [75...gk, 6h...14], "hi")

```

heyanon

用以太坊账户列表的身份发布帖子，通过 zk 验证消息是否为来自于集合成员的有效签名。



spec: heyanon.xyz

注册阶段



vivekab 用公钥加入: 0x414e...

lsankar 用公钥加入: 0x15a9...



Database

vivekab	->	0x414e...
lsankar	->	0x15a9...

spec: heyanon.xyz

发消息阶段



Database

vivekab -> 0x414e...

lsankar -> 0x15a9...

vivekab 生成zk证明:

component prove:

public input pks[n]
public input message

private input pk
private input signature_of_message

$(pk - pks[0]) * (pk - pks[1]) * \dots == 0$

verify(signature_of_message, pk, message) == 1

prove([0x414e..., 0x15a9...], "hi", 0x414e..., sign("hi", vivek_secret_key))

spec: heyanon.xyz

发消息阶段



Database

merkle_root -> 0x59...

vivekab 生成zk证明:

component prove:

public input merkle_root
public input message

private input pk
private input signature_of_message

private input merkle_path[logn]
hash(hash(pk, merkle_path[0]), merkle_path[1])... == merkle_root

verify(signature_of_message, pk, message) == 1

prove(merkle_root=0x59..., "hi", 0x414e..., sign("hi", vivek_secret_key), merkle_path)

vivekab -> 0x414e.. lsankar -> 0x15a9...

nullifiers 的用途

Group Membership (无 nullifiers)

- zkmessage -> hash(secret), set membership
- heyanon -> signature verification, merkle paths

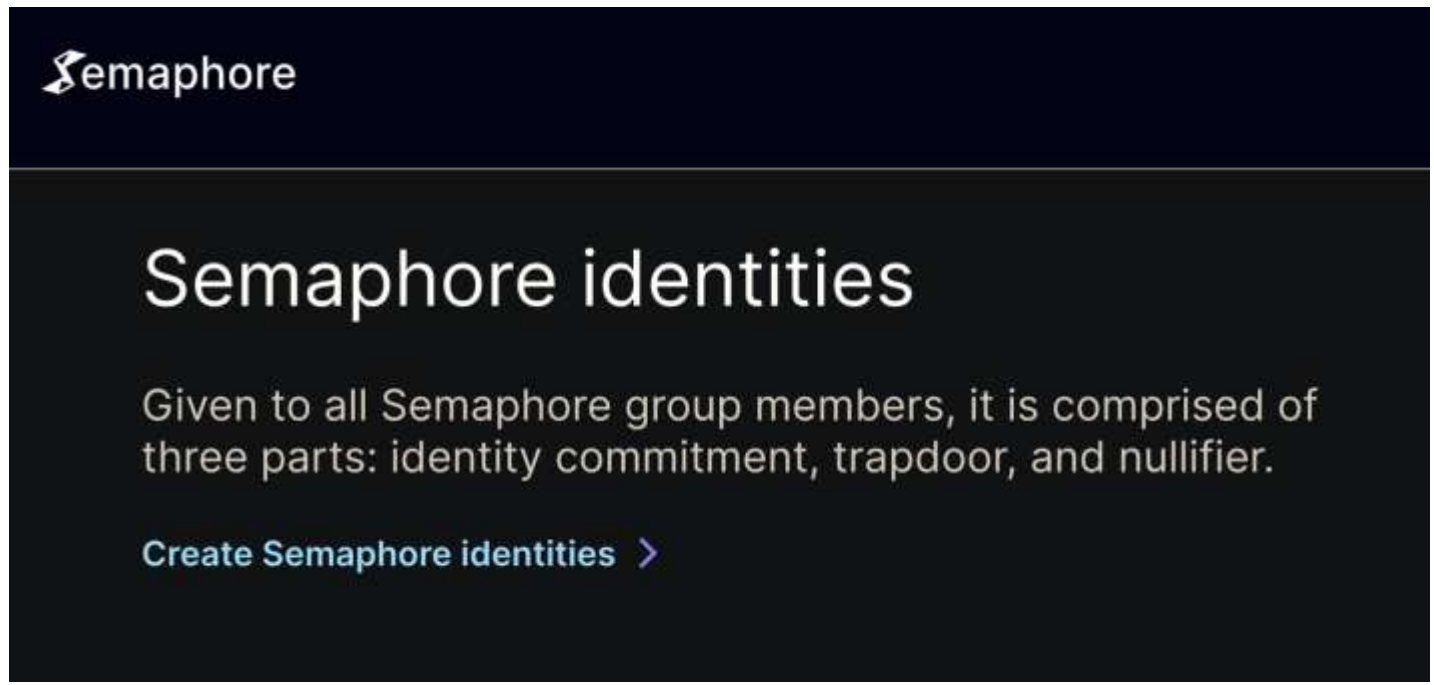
在有些情况下，希望使用户无法匿名地执行两次操作。

Group Membership (有 nullifiers)

- semaphore -> hash(secrets), merkle paths, **nullifiers**
- tornado cash -> hash(secrets), merkle paths, **nullifiers**

semaphore

证明哈希列表中的成员身份，并包含一个无效器（nullifier）。



spec: semaphore

注册阶段



Database

merkle_root -> 0x59...

weijie -> hash(n1, t1)

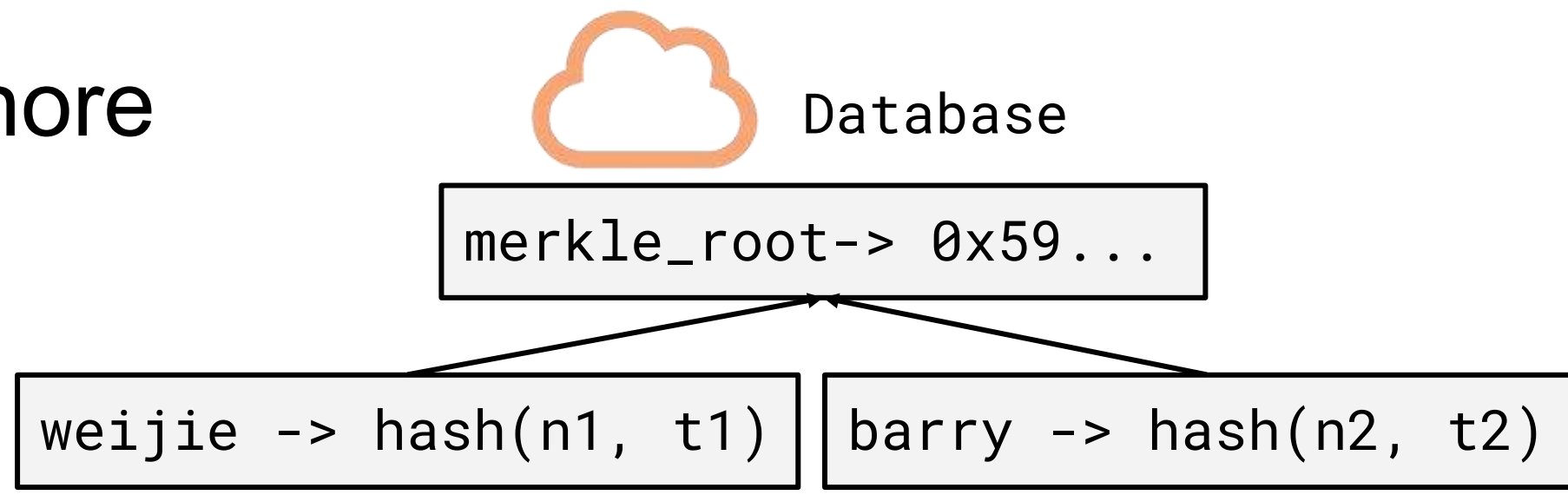
barry -> hash(n2, t2)

 weijie 以哈希身份加入: hash(nullifier, trapdoor).
nullifier, trapdoor 是私有的.

 barry 以哈希身份加入: hash(nullifier2, trapdoor2).
nullifier2, trapdoor2 是私有的.

spec: semaphore

发消息阶段



 weijie -> "Yes!" + zk proof +
hash(nullifier1, "Does pineapple belong on pizza?")

 barry -> "No!" + zk proof +
hash(nullifier2, "Does pineapple belong on pizza?")

spec: semaphore

发消息阶段



<https://github.com/semaphore-protocol/semaphore/blob/main/packages/circuits/semaphore.circom>

Database

merkle_root -> 0x59...

weijie -> hash(n1, t1)

barry -> hash(n2, t2)

weijie 生成zk证明:

component prove:

```
public input merkle_root
public input message # "Yes"
```

```
public input external_nullifier # hash("Does pineapple belong on pizza")
```

```
private input n1 # identity_nullifier
```

```
private input t1 # identity_trapdoor
```

```
private input merkle_path[logn]
```

```
merkle_leaf <== hash(n1, t1)
```

```
hash(hash(merkle_leaf, merkle_path[0]), merkle_path[1])... == merkle_root
```

```
public output nullifier_hash
```

```
nullifier_hash <== hash(n1, external_nullifier)
```

```
prove(merkle_root=0x59..., "Yes", "Does...", nullifier1, trapdoor1, merkle_path)
```

spec: semaphore

发消息阶段



<https://github.com/semaphore-protocol/semaphore/blob/main/packages/circuits/semaphore.circom>

Database

merkle_root -> 0x59...

weijie -> hash(n1, t1)

barry -> hash(n2, t2)

Nullifier Hashes

nullifier_hash1

weijie 发送zk证据:

```
{proof_a, merkle_root, message = "Yes", nullifier_hash1}
= prove(merkle_root=0x59..., "Yes", "Does...", nullifier1, trapdoor1, merkle_path)
```

```
{proof_b, merkle_root, message = "No", nullifier_hash1}
= prove(merkle_root=0x59..., "No", "Does...", nullifier1, trapdoor1, merkle_path)
```

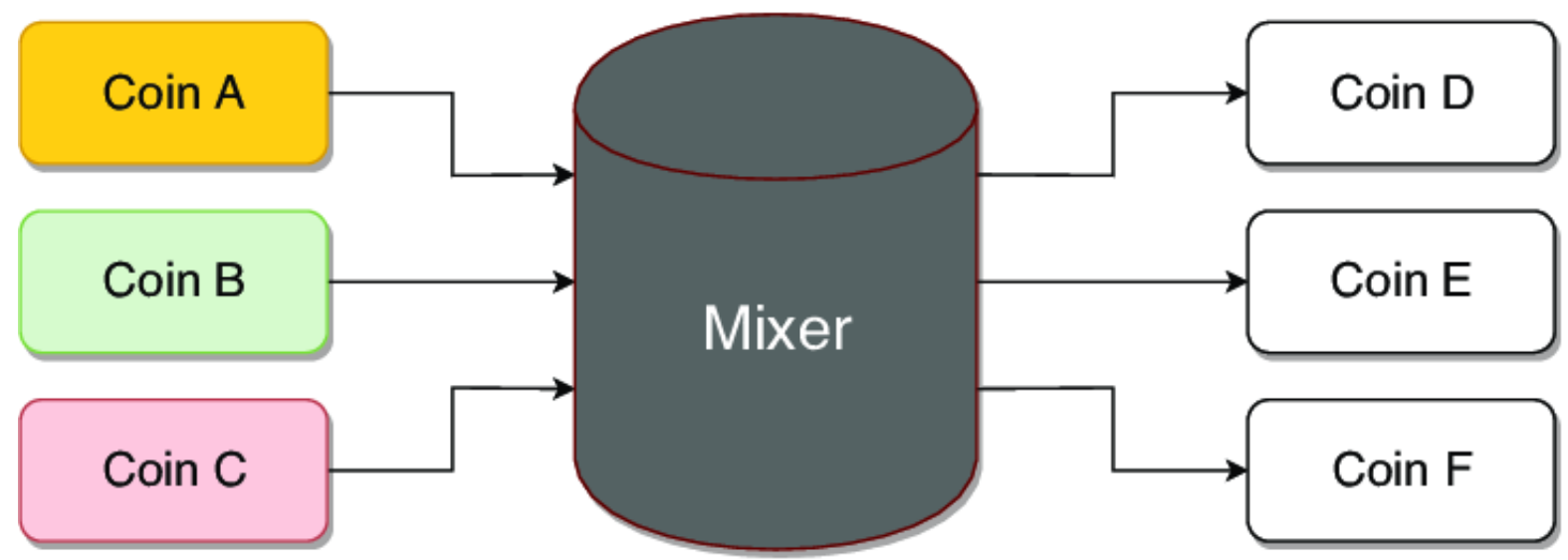
tornado.cash

通过每次提款附加一个zk成员身份证明和一个nullifier，向匿名账户发送资金。



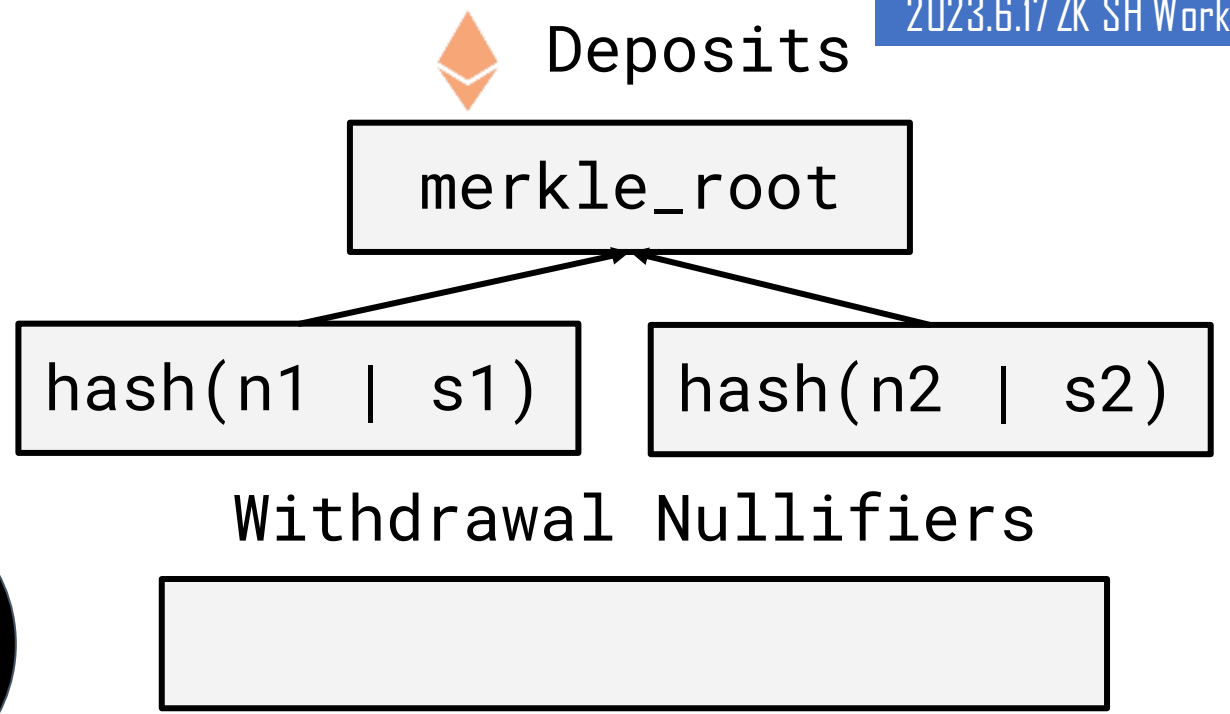
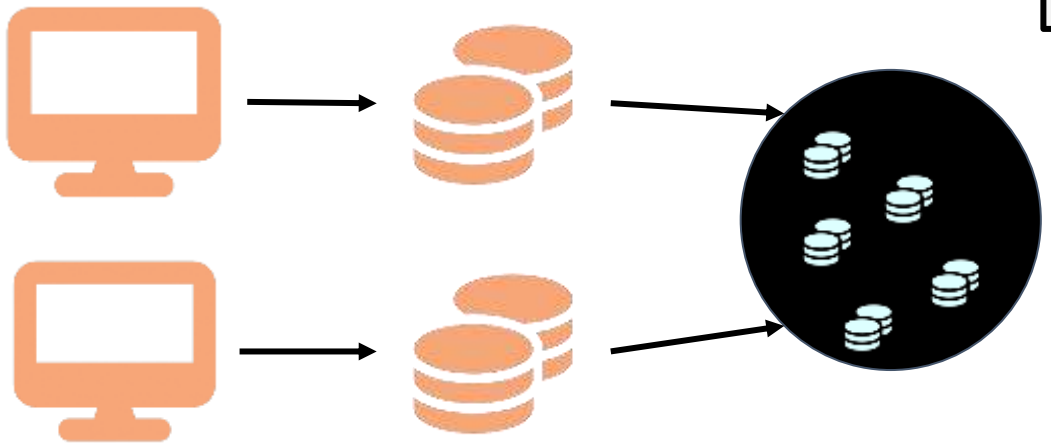
混币器

Coin Mixer



spec: tornado cash

存款阶段



alexis.eth 发送 1 eth 并 $\text{hash}(\text{nullifier1} \mid \text{secret1})$

betty.eth 发送 1 eth 并 $\text{hash}(\text{nullifier2} \mid \text{secret2})$

spec: Tornado Cash

提款阶段

alexis 生成zk证明:



<https://github.com/tornadocash/tornado-core/blob/master/circuits/withdraw.circom>

```
component prove:
  public input merkle_root
  public input recipient_pk
  public output nullifier_hash

  private input nullifier
  private input secret
  private input merkle_path[logn]

  merkle_leaf <== hash(nullifier, secret)
  hash(hash(merkle_leaf, merkle_path[0]), merkle_path[1])... == merkle_root

  public output nullifier_hash
  nullifier_hash <== hash(nullifier)

  prove(merkle_root=0x59..., recipient_pk=0x7ab89..., nullifier1, secret1, merkle_path1)
```

spec: Tornado Cash

提款阶段

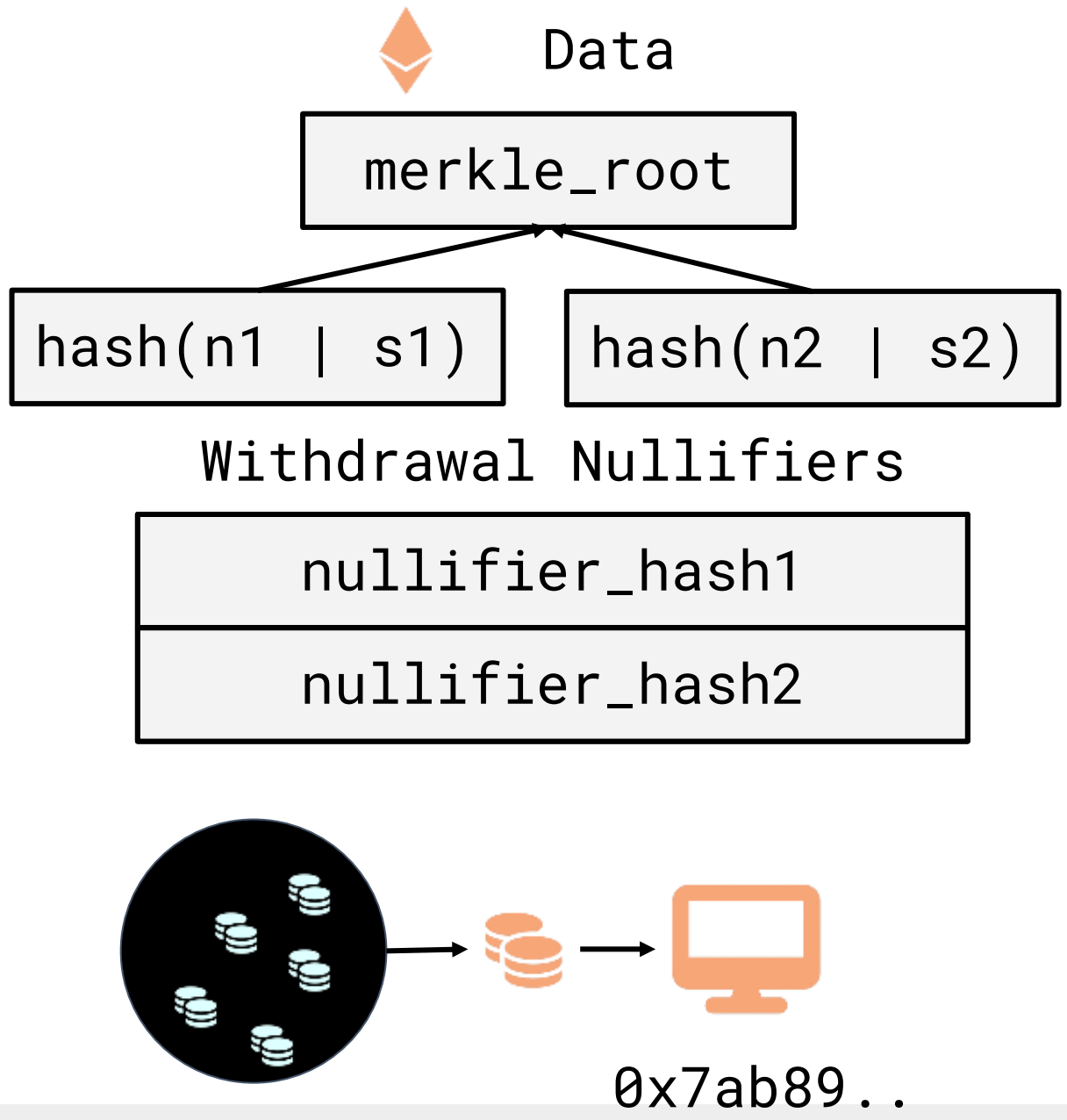
alexis 生成zk证明:
betty 生成zk证明:



{zk_proof, merkle_root, recipient, nullifier_hash}

链上合约

- 验证zk证据
- 验证merkle root是匹配的
- 验证nullifier_hash并没有在列表
- 将nullifier_hash加入到列表中
- 将款项发送给接收者



可以做到更多吗?

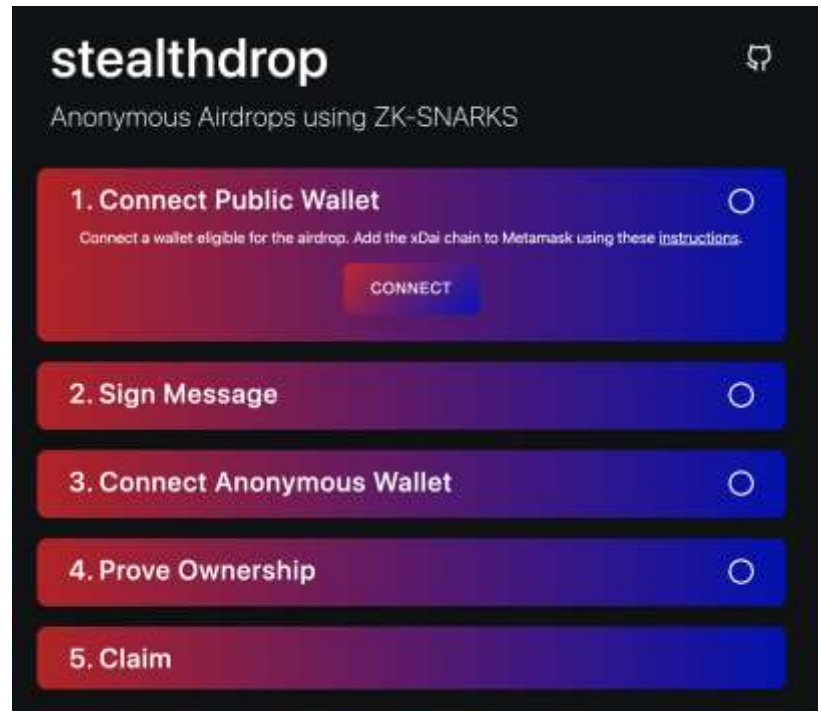
heyanon -> 签名验证

hash(secrets), merkle paths, nullifiers

如何把这两种混合呢?

stealthdrop (以及为什么它有漏洞)

通过每次提款附加一个zk成员身份证明和一个nullifier, 向无关联账户 (unlinked account) 申领空投。

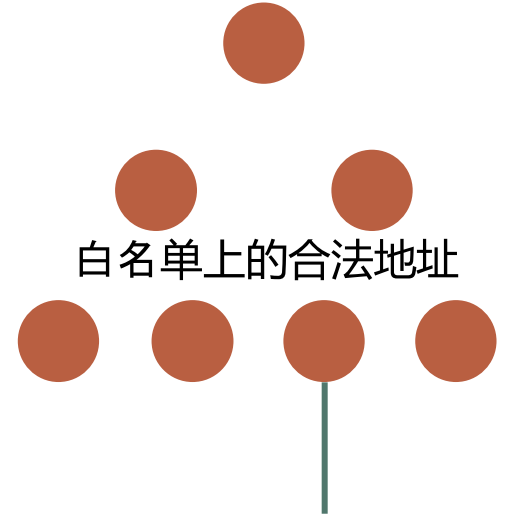


stealtdrop

流程

空投 池子

Merkle Root



ZK:

adhyyan.eth

Nullifier 列表

hash(sig)



coin

0xad829...

spec: stealthdrop

提款阶段

nalin 生成zk证明:



Airdrop List

merkle_root

nalin.eth

adhyyan.eth

Withdrawal Nullifiers

nullifier_hash=7ba93...

component prove:

```
public input merkle_root
public input message
```

```
private input pk
private input signature_of_message
```

```
private input merkle_path[logn]
```

```
hash(hash(pk, merkle_path[0]), merkle_path[1])... == merkle_root
```

```
verify(signature_of_message, pk, "stealthdrop out") == 1
```

```
public output nullifier_hash
nullifier_hash = hash(signature_of_message)
```

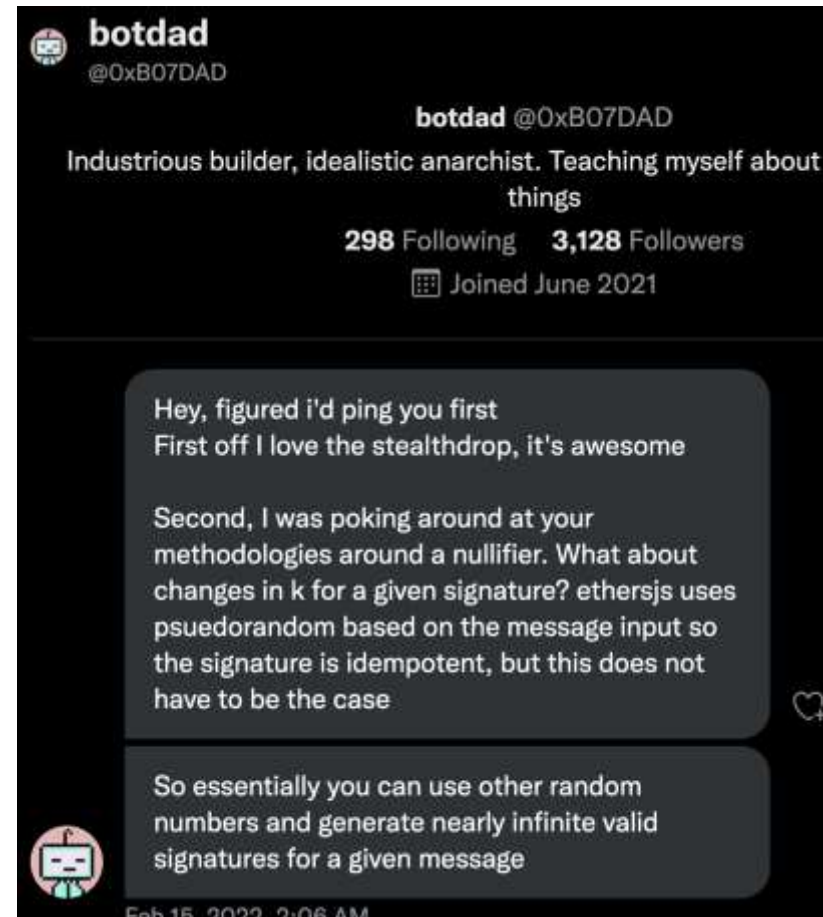
```
prove(merkle_root=0x59..., nalin.eth., sign("stealthdrop out", nalin_sk), merkle_path)
```

漏洞利用

问题：非确定性签名

(但确定性ECDSA呢?)

解决方案：Priv2pub或...



匿名nullifier的特性

- 唯一性的
- 确定性的
- 不需要私钥即可验证
- 非交互式的（不像tornado.cash或semaphore）

一些不能工作的想法

确定性 ECDSA 签名

$\text{hash}(\text{message}, \text{public key})$

$\text{hash}(\text{message}, \text{secret key})$

$\text{hash}(\text{message}, \text{public key})^{\text{secret key}} \rightarrow \text{DDH-VRF!}$

我们需要一个基于用户私钥的确定性函数，
该函数可以仅通过用户的公钥进行验证，
并保持其匿名。

解决方法：在以太坊上部署确定性签名。

方案

已知公私钥对(sk , $pk = g^{sk}$), 公共信息 m

Signature:

public: $hash[m, pk]^{sk}$ <-- nullifier

private: $c = hash2(g, pk, hash[m, pk], hash[m, pk]^{sk}, g^r, hash[m, pk]^r)$

$s = r + sk * c$

$pk = g^{sk}$

g^r [optional output]

$hash[m, pk]^r$ [optional output]

user proves in SNARK:

$g^{[r + sk * c]} / (g^{sk})^c = g^r$

$hash[m, g^{sk}]^{[r + sk * c]} / (hash[m, pk]^{sk})^c = hash[m, pk]^r$

$c = hash2(g, g^{sk}, hash[m, g^{sk}], hash[m, pk]^{sk}, g^r, hash[m, pk]^r)$

pk is in anonymity set (merkle proof)

spec: new stealthdrop

提款阶段

nalin 生成zk证明:



Airdrop List

merkle_root

nalin.eth

adhyyan.eth

Withdrawal Nullifiers

plume_signature_of_message

```
component prove:
  public input merkle_root

  private input pk, c, s
  public input plume_signature_of_message

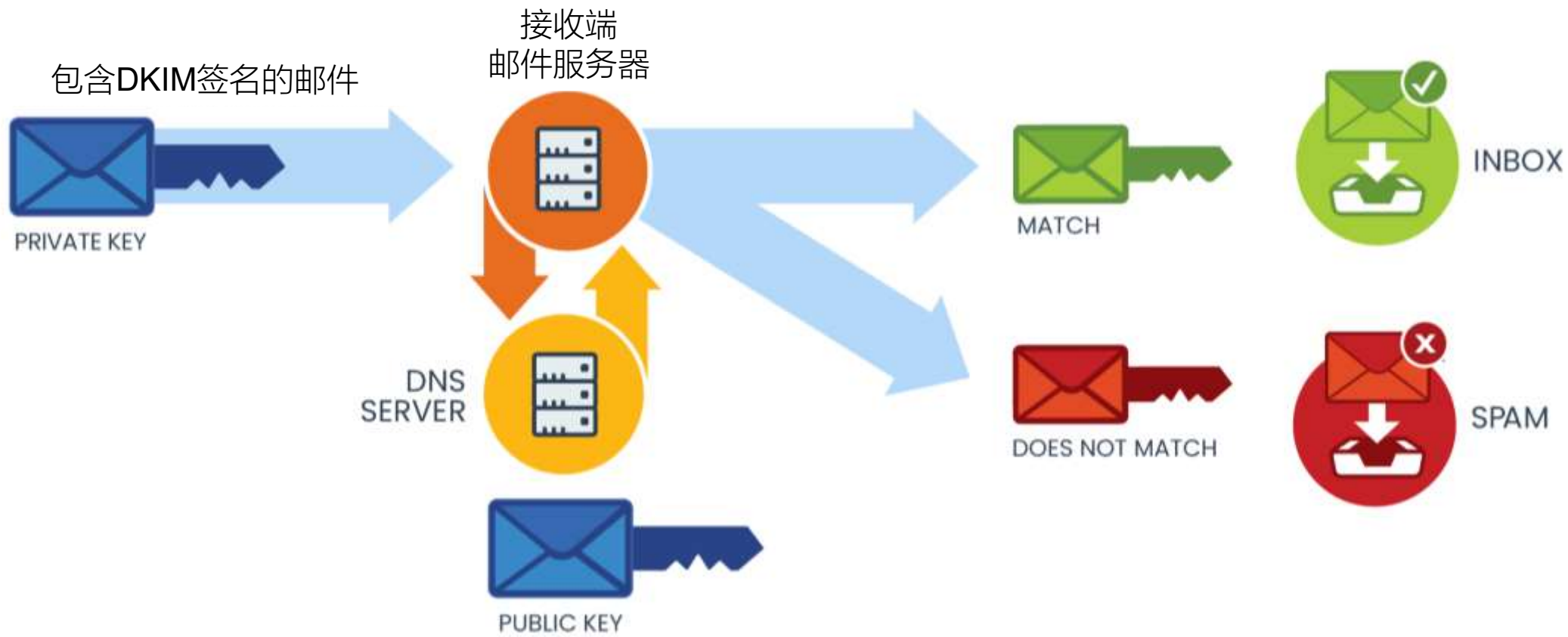
  private input merkle_path[logn]

  hash(hash(pk, merkle_path[0]), merkle_path[1])... == merkle_root

  verify(plume_signature_of_message, pk, c, s, "stealthdrop out") == 1
  # nullifier == plume_signature_of_message

prove(merkle_root=0x59..., nalin.eth., c, s, plume_signature_of_message, merkle_path)
```

DKIM认证过程




DKIM 方案

`rsa_sign(sha256(from:<>, to:<>, subject:<>, <body>,...), @mit.edu key)`

用户y只需要提供任何一个例如“来自x@twitter.com发送给y@mit.edu的电子邮件头”，利用零知识证明的隐藏能力，即使是邮件服务器或按键跟踪器，也无法确定是谁发送的。

一些典型问题：

- gmail.com不会对自身电子邮件进行签名（直到2016年才开始检查DKIM）
- hotmail.com忘记包括收件人字段 
- mit.edu在outlook中表现良好！
- perfect.blue（即自定义邮件服务器）使用1024位rsa。

证明

- ZK证明公开部分：
 - 发件人域名（和/或收件人域名）
 - RSA模数
 - 掩码电子邮件正文
- ZK证明私有部分：
 - 来自邮件服务器的DKIM签名
 - 预哈希消息数据的长度
 - 原始消息数据
- ZK电路检查：
 - sha256和RSA均进行验证
 - 发件人/收件人域在消息中结构良好[正则表达式]
- 合约检查：
 - 发件人域名= verify@twitter.com
 - 声称的RSA密钥= DNS获取/缓存的RSA密钥

值得注意的

- 共有财产的悲剧
 - 个人“花费”机构声誉
 - 发布前没有零知识证明的公共身份
- 可否认性?
 - mit.edu/~specter/blog/2020/dkim/
- nullifier的权衡
 - 如果您想避免人们在链上声明多个身份/出售身份，则必须允许发送邮件服务器对某人进行去匿名化
- 旋转公钥
- 密送抄送 (BCCs)

zk-email 应用

- **Twitter**验证电子邮件 => 证明您拥有特定的**Twitter**账户以创建匿名集合
- 匿名KYC / 人格证明 => 证明您收到了{AirBNB, Coinbase, Robinhood}的有效KYC电子邮件, 以证明您是真正的人类
- 向**arxiv**论文捐款 => 证明您拥有发送**arxiv**论文的电子邮件, 可以领取资金

其他 zk-email 应用

- **Robinhood**交易电子邮件通知 => 证明您拥有X数量的Y股票, 证明您在交易中赚取了\$X
- **Chase**银行账户余额电子邮件 => 证明您的银行账户中有\$X
- **Spotify**顶级粉丝电子邮件 => 证明您是某位艺术家的顶级粉丝, 以获得特殊待遇和VIP门票
- **Twitter**直接消息电子邮件通知 => 证明您收到了来自特定帐户的特定DM

Research/theoretical work

- 客户端证明
 - 为特定操作定制的证明堆栈
 - 用于验证ECDSA签名（以及通常使用secp256k1进行ECC操作），正在使用secq曲线开发spartan-ecdsa和halo2-ipa
 - 将现代证明堆栈从主要的Rust转换和优化为WASM，以进行浏览器和移动证明
 - 使用递归/聚合方案来组合不同的证明
 - Nova、plonky2、Zigzagoon

Research/theoretical work

- 通过MPC进行私人委托
 - 有些证明将会因为太大而在用户设备上长时间无法证明
 - 下一个最好的解决方案是将证明私下委托给另一台服务器
 - 可以使用FHE进行某些操作，但这似乎还需要更长时间的研究
 - 伯克利和斯坦福的积极研究正在实现一种方案，其中您的证人/私人数据被多个方共享并分别计算
 - 需要将其投入生产并得到使用!

Writing/lobbying work

- 将nullifiers投入生产!
 - 将此API构建到钱包中
- 在API端点上获取更多签名
 - 可以使用公证解决方案，但最终最好的设置是实际数据源签署所有内容
- 让更多的网站使用SXG
 - 然后，所有HTML都由网站的公钥签名，然后可以验证某些文本来自nytimes.com（用于新闻）或nfl.com（用于体育比分）
- 教育/准备机构更多的ZK和匿名性

Product work

- heyanon隐身衣
 - 即将推出的产品，将机器人添加到各种群聊中（Discourse、Discord、Telegram、WhatsApp），允许该群聊的成员注册并发送匿名消息
 - 可以将进一步的声誉附加到这个“隐身衣”（群组管理员，在您的discord中的角色）
 - 将使用FaceID / TouchID进行身份验证，使用WebAuthn技术
- heyanon面对面
 - 在一些会议上部署了heyanon（请查看@DevconAnon!）
 - 对于最后一节课，我将使用ZK构建一个匿名反馈表格供人们使用！

Product work

- 组织的匿名性
 - 与DAO、学校和俱乐部合作，为使用案例设置具有适当的调节和匿名性保证的内部匿名通道
- 艺术表现
 - 匿名艺术家公会共同创作（写作、艺术等），但他们不会透露自己的身份
 - 同时他们还可以以一个群体进行发声
- 更好的治理
 - 可以基于链上投票设置派别/意识形态
 - 派别变得更有意义和清晰定义