

# 门限签名

冀甜甜

日期：2023年3月26日

# 系列问题

---

- BLS、SSS、门限、多签四个关键词之间的关系是什么？
- 多方安全计算与门限签名的关系是什么？
- TSS 与 DVT、DPKI 的关系是什么？能不能说TSS是他们的底层基础设施呢？
- BLS 签名与ECDSA 签名的关系？
- 相关产品：ZenGo,其初衷是不是要解决私钥安全问题呢？
- Gnosis safe使用的是多重签名吗？
- Ronin多签安全问题：Axie DAO Validator将他们的多重签名借给了 Sky Mavis  
→ 怎么借的？是将私钥借出了吗？
- LAZARUS GROUP APT组织是不是正在往私钥盗窃的方向发展？
- 基于MPC的门限签名:BitiZen钱包门限签名方案？ZK可信设置？DVT？
- 签名安全本质上是否完全等价于私钥安全？

# 提纲

一

多签、聚合与门限签名

二

Shamir秘密共享、多签与门限签名

三

BLS n-of-m门限签名

四

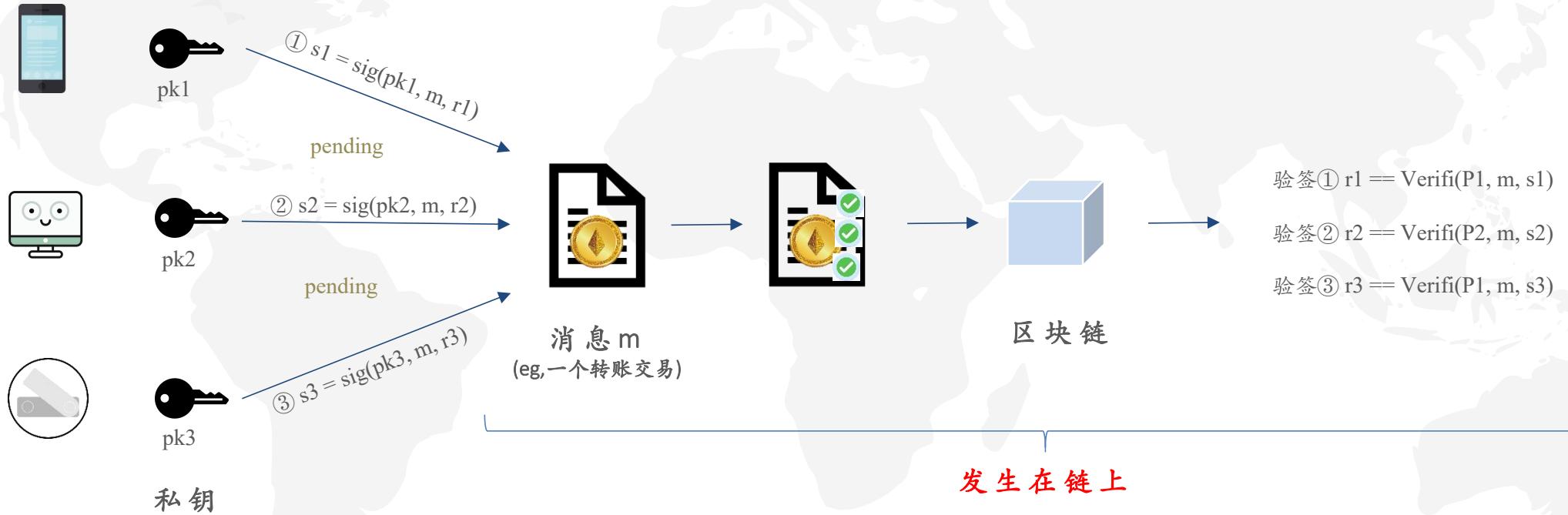
Gnosis Safe的签名方案 – 多签非门限

五

DVT技术与门限签名技术有何关联？

# 多签、聚合与门限签名

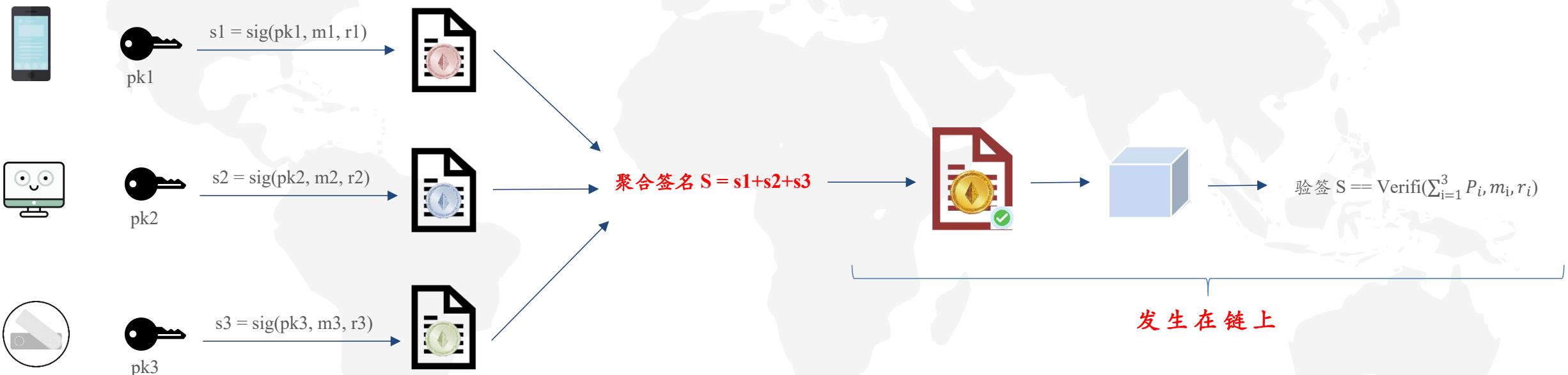
- 多重签名：多设备（/参与方/私钥）共同签署一个消息  $m$ （/ $\text{hash}(m)$ ）
- 聚合签名：多设备签署多个消息  $m$
- 门限签名： $m$ -of- $n$ 个设备共同签署一个消息  $m$



- Pending 表示参与方可以离线
- 每个参与方的签名都需要单独验证 ( $\rightarrow$  Schnorr 签名可以实现一次验证)
- 参与方其中之一丢了私钥，会导致交易签名无法完成

# 多签、聚合与门限签名

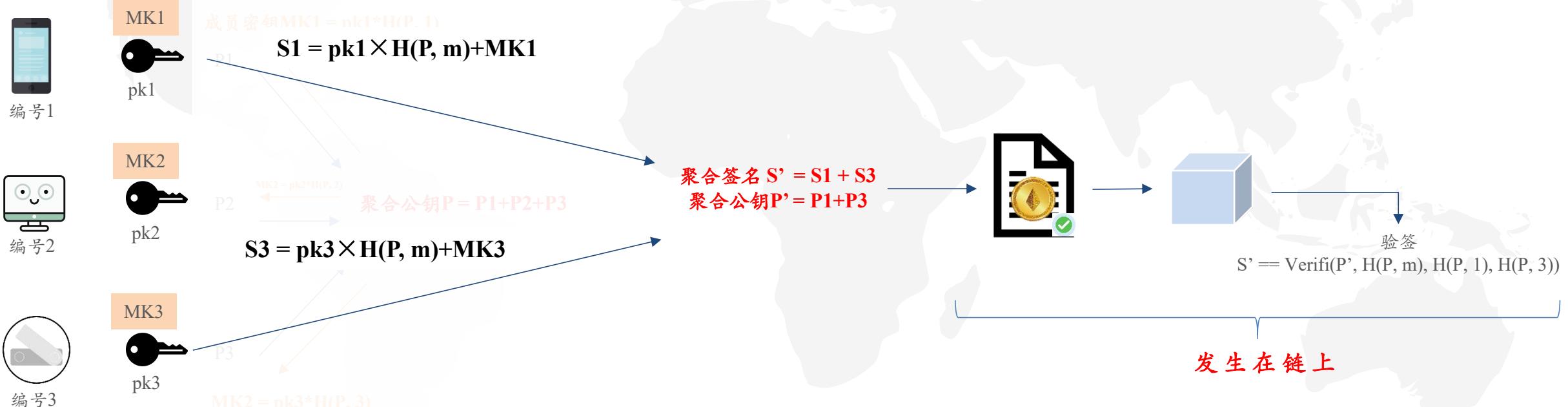
- 多重签名：多设备（/参与方/私钥）共同签署一个消息 $m$ （/ $\text{hash}(m)$ ）
- 聚合签名：多设备签署多个消息 $m$
- 门限签名： $m$ -of- $n$ 个设备共同签署一个消息 $m$



- 参与方之间可以没有关系，也可以共同签名一笔交易，也可以适用一人签署多个交易的场景
- 存储任务减轻：链上不再需要存储所有签名，仅需要存储一个聚合签名即可
- 计算任务减轻：将所有签名验证方程相加，可节省一定的计算能力，节约gas费（有时可以搭配聚合公钥 $P=P1+P2+P3$ ）

# 多签、聚合与门限签名

- 多重签名：多设备（/参与方/私钥）共同签署一个消息  $m$ （/ $\text{hash}(m)$ ）
- 聚合签名：多设备签署多个消息  $m$
- 门限签名： $m$ -of- $n$ 个设备共同签署一个消息  $m$



- 需要一个额外的设置阶段来生成成员密钥  $MK$
- 成员密钥是  $n$ -of- $n$  的签名，用来证明是门限签名的有效参与者
- 要求门限签名参与方实时在线
- $H()$  函数具有非线性，无法通过链上存储的聚合签名获取任何关于成员签名的信息

# 多签、聚合与门限签名

- 多重签名：多设备（/参与方/私钥）共同签署一个消息  $m$ （/ $\text{hash}(m)$ ）
- 聚合签名：多设备签署多个消息  $m$
- 门限签名： $m$ -of- $n$ 个设备共同签署一个消息  $m$

	私钥数量	所需消息数量	公开的公钥数量	所需签名者数量	公开的签名数量
<b>ECDSA</b>	简单多签名	$n$	1	$n$	$n$
<b>Schnorr</b> <b>Musig</b> <b>BLS</b>	多签名	$n$	1	1	$n$
	聚合签名	$n$	$1^*$	1	$n$
	门限签名**	$n$	1	1	$m < n$

(\*) 在此案例中，消息可以是完全不同的，最多可以有  $n$  条不同的消息

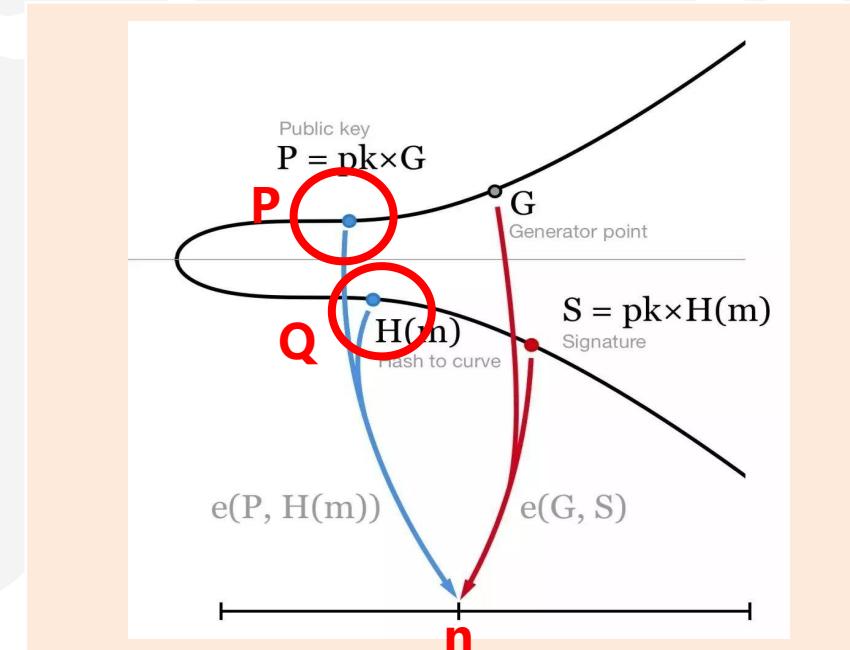
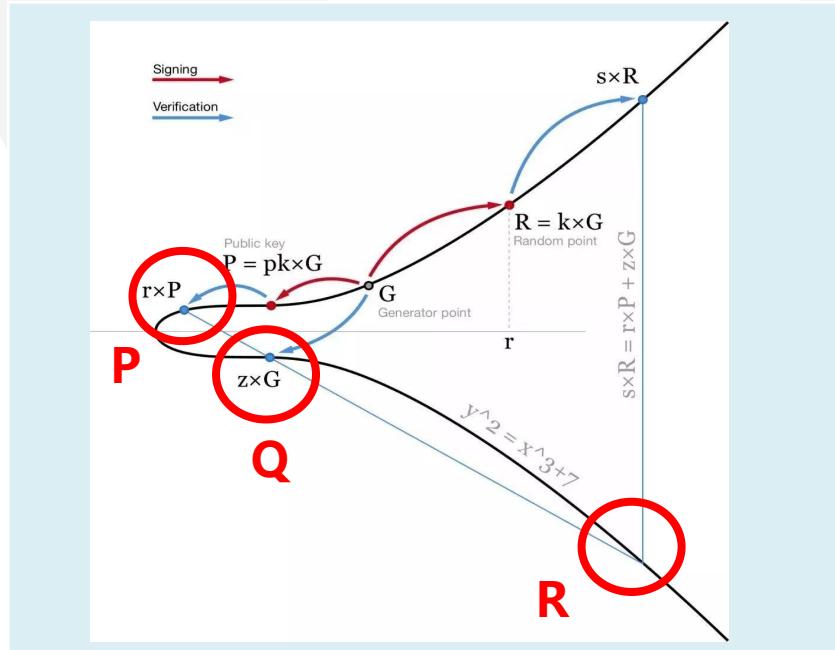
(\*\*) 假设是  $m-n$  的门限方案

# 体系化/家族化

➤ 从验签原理上可以归纳为两类：

(1) 利用 “ $P + Q + R = 0$ ” 的性质：ECDSA、Schnorr、Musig签名算法

(2) 利用 “ $e(P, Q) \rightarrow n$ ” 与 “ $e(x \times P, Q) = e(P, x \times Q)$ ” 的性质：BLS签名算法



简单、存储小，但这种验证效率并不比左侧高。

# 提纲

一

多签、聚合与门限签名

二

Shamir秘密共享、多签与门限签名

三

BLS n-of-m门限签名

四

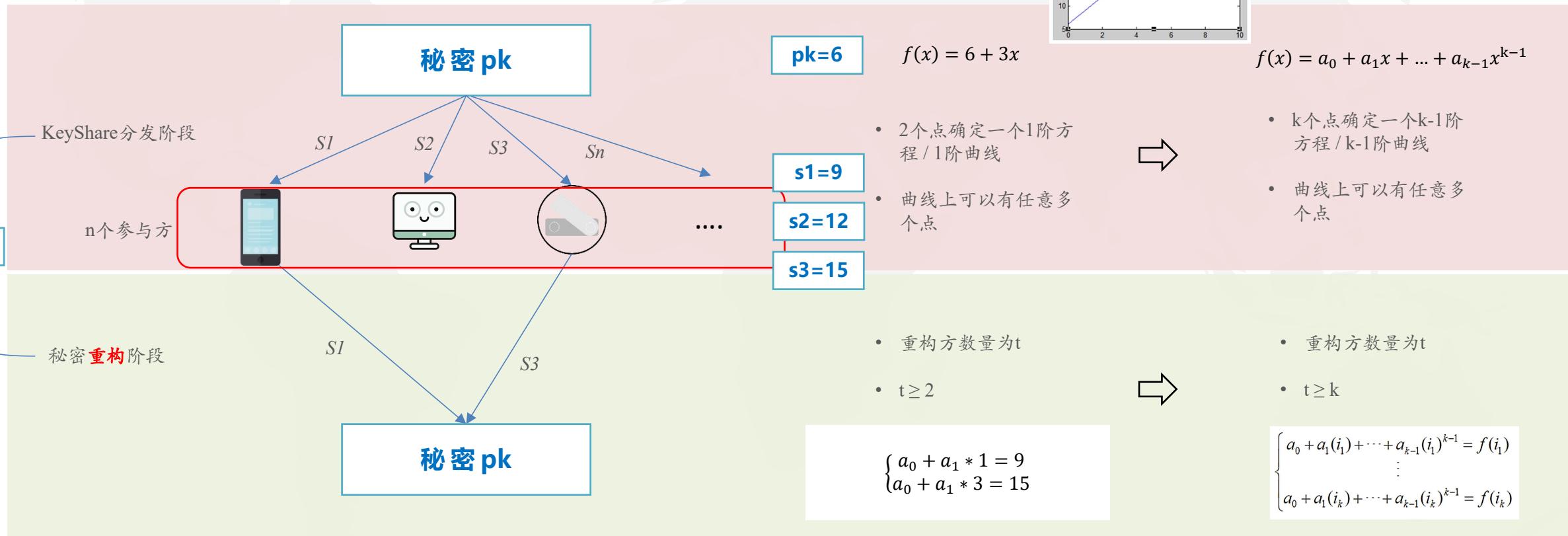
Gnosis Safe的签名方案 – 多签非门限

五

DVT技术与门限签名技术有何关联？

# Shamir秘密共享、多签与门限签名

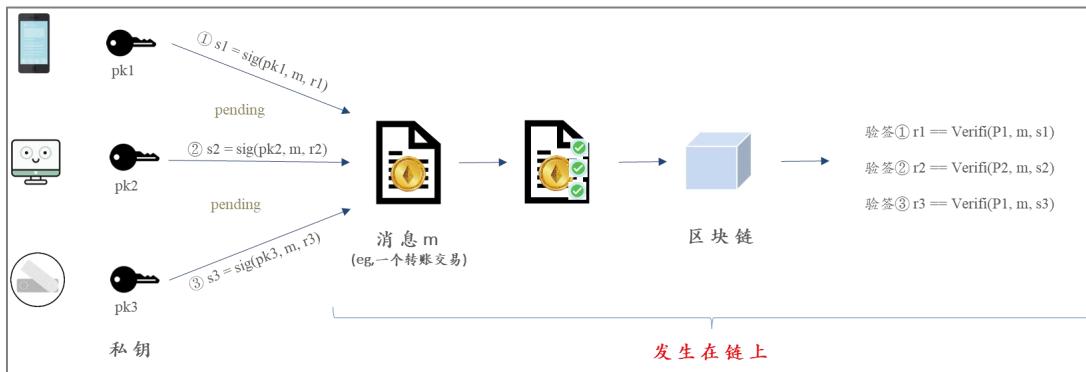
- Shamir秘密共享 (SSS)：需要重构私钥，存在dealer作恶、私钥窃取的安全问题
- 多重签名：需要公开多个签名信息
- BLS门限签名：无需重建私钥、无需将多个签名信息公开



- 秘密共享的发展路线: SSS → VSS → DKG(无dealer), SSS=Shamir秘密共享, VSS=Verifiable Secret Sharing, DKG=Distributed Key Generation

# Shamir秘密共享、多签与门限签名

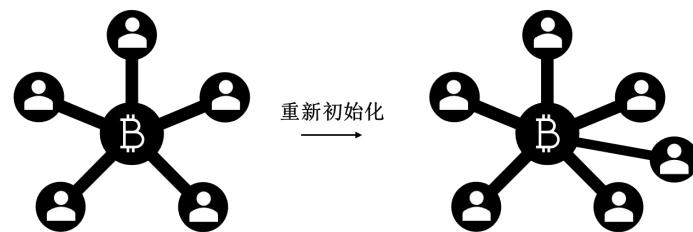
- Shamir秘密共享 (SSS)：需要重构私钥，存在dealer作恶、私钥窃取的安全问题
- 多重签名：需要公开多个签名信息，存储和隐私性上均处于弱势
- BLS门限签名：无需重建私钥、无需将多个签名信息公开，可刷新keyshare



在链上，对所有签名信息进行依次验签

◆ 公钥 PK1 PK2 PK3 PK4 PK5 PK6 PK7 PK8 PK9 PK10 PK11 PK12  
◆ 多种签名 S1 S2 S3 S4 S5 S6 S7 S8 S9 S10 S11 S12

## 1. 访问结构无法调整

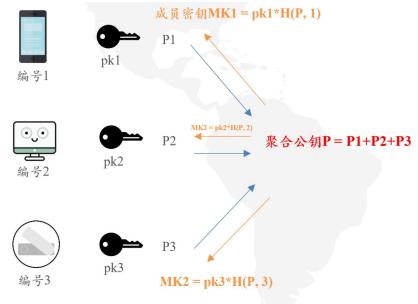


## 2. 效率问题

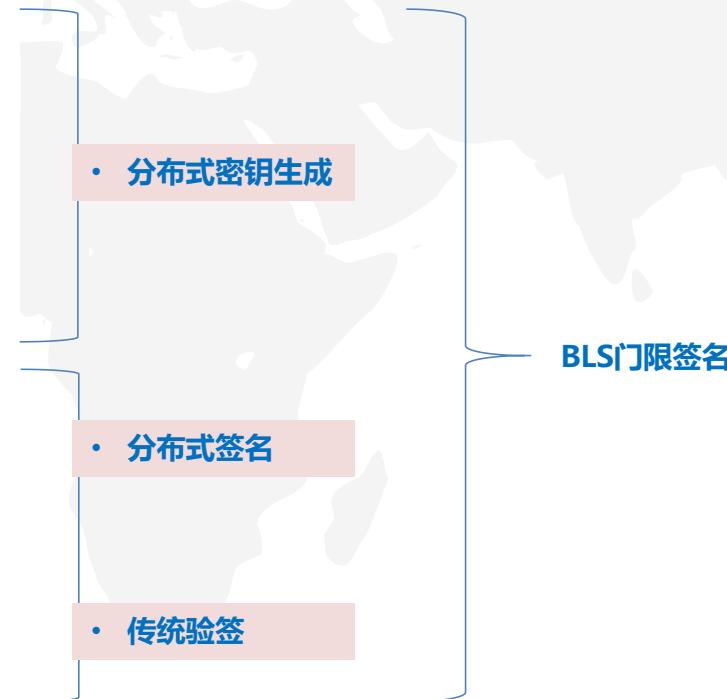
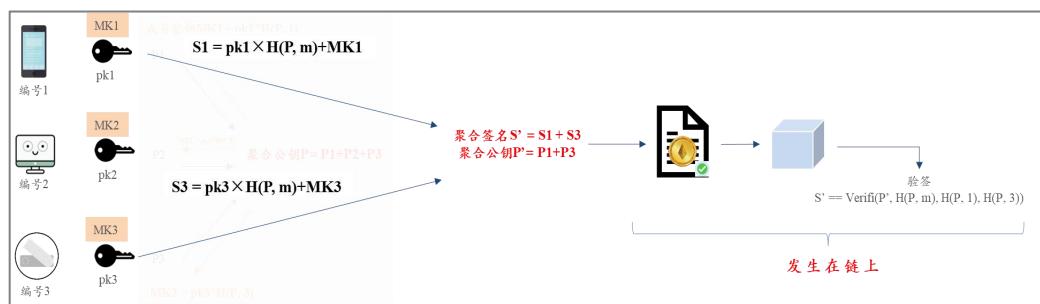
3. 无匿名性：签名信息的公开，使得任何人都可以看到它具有不寻常的安全机制，因此可以更容易地跟踪。

# Shamir秘密共享、多签与门限签名

- Shamir秘密共享 (SSS)：需要重构私钥，存在dealer作恶、私钥窃取的安全问题
- 多重签名：需要公开多个签名信息，存储和隐私性上均处于弱势
- BLS门限签名：无需重建私钥、无需将多个签名信息公开，可刷新KeyShare



• MK为每个参与者互不知晓的私钥碎片

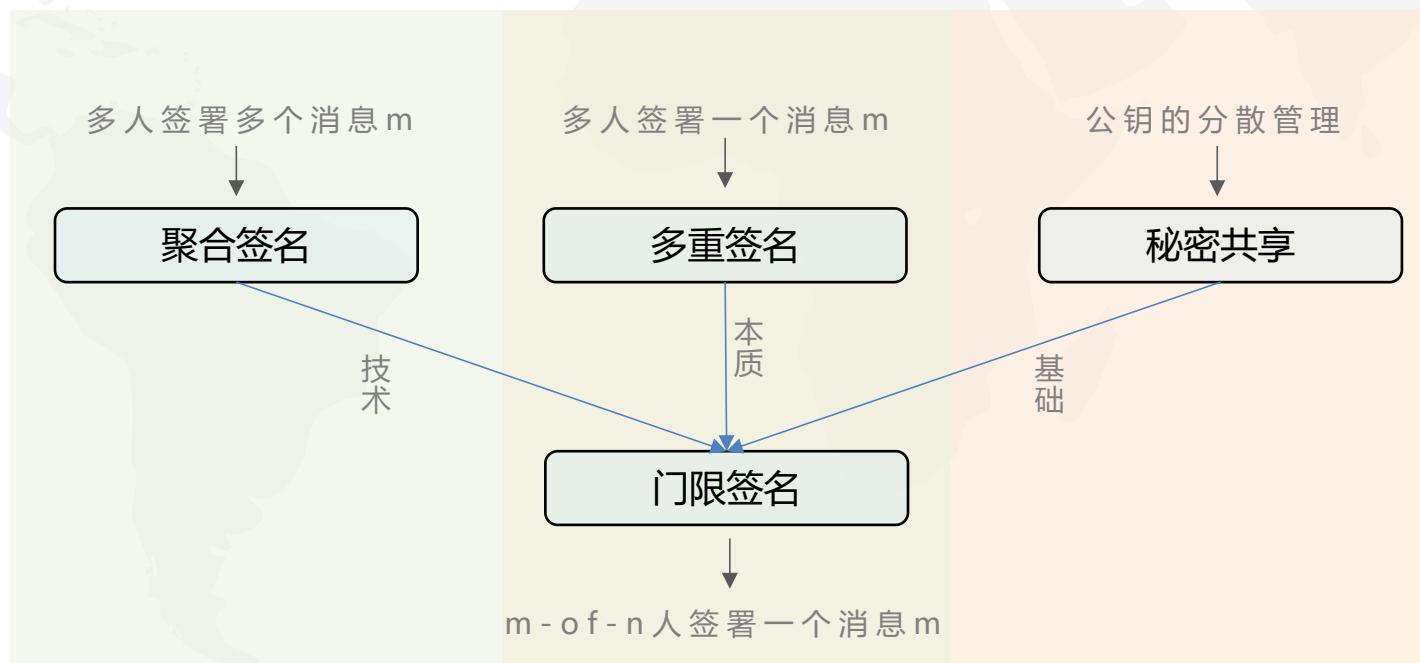


对于广义的门限签名而言：  
第一步中的“分布式密钥生成”并不特指DKG，也可以采用Shamir秘密共享来实现。

注：SSV就是采用SSS算法实现的分布式密钥生成。

# 体系化归纳

- 聚合签名、多重签名、秘密共享不存在关联关系，所提出的背景和要解决的问题不同
- 三者与门限签名的关系：
  - 门限签名：本质上仍属于多重签名的范畴
  - 秘密共享：最初是为解决密码学公钥的分散管理提出的，后成为门限密码学的基础，门限签名中的分布式密钥生成技术本质上属于秘密共享的范畴
  - 聚合签名：仅将其理解为门限签名实现过程中的一种技术手段



# 提纲

一

多签、聚合与门限签名

二

Shamir秘密共享、多签与门限签名

三

BLS m-of-n门限签名

四

Gnosis Safe的签名方案 – 多签非门限

五

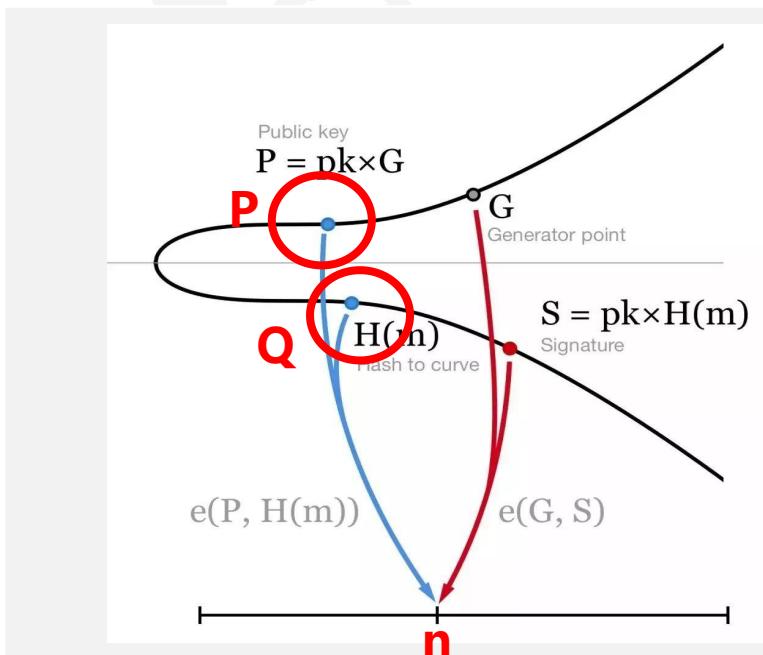
DVT技术与门限签名技术有何关联？

# BLS m-of-n门限签名

- 示例：构建3个不同设备上的2-of-3多重签名方案（为1笔交易/消息共同签名）

➤ 已知信息 → ①设置阶段 → ②签名 → ③验签

- 椭圆曲线G（G称为椭圆曲线的生成器检查点）
- 消息m → Hash(m)
- 私钥pk1、pk2、pk3



- $e(P, Q) \rightarrow n$
- $e(x \times P, Q) = e(P, x \times Q)$

$$\begin{aligned} & e(a \times P, b \times Q) \\ &= e(P, ab \times Q) \\ &= e(ab \times P, Q) \\ &= e(P, Q)^{ab} \end{aligned}$$

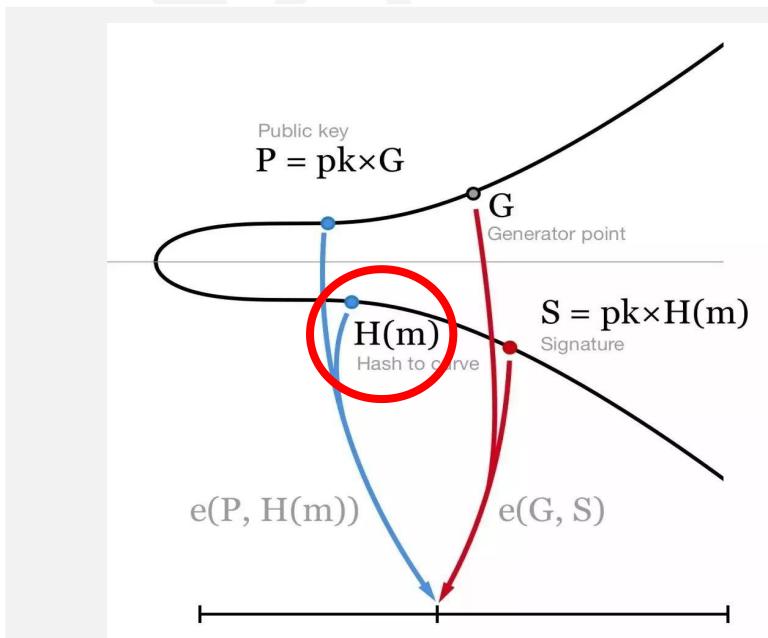
$$e(2 \times P, Q) = e(P, Q) \times e(P, Q)$$

# BLS m-of-n门限签名

- 示例：构建3个不同设备上的2-of-3多重签名方案（为1笔交易/消息共同签名）

➤ 已知信息 → ①设置阶段 → ②签名 → ③验签

- 椭圆曲线G（G称为椭圆曲线的生成器检查点）
- 消息m → Hash(m)
- 私钥pk1、pk2、pk3



- 单个设备签名与验签时：

## ① 签名

计算公钥  $P = pk \times G$   
计算签名  $S = pk \times H(m)$

消息哈希到曲线

## ② 验签

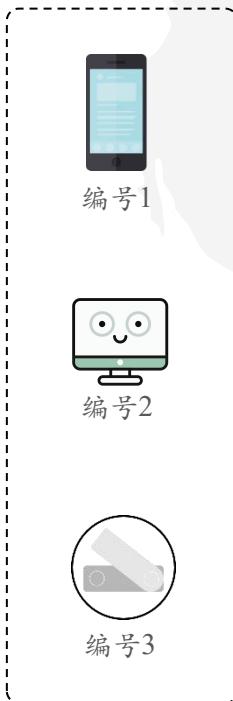
$e(P, H(m)) == e(G, S)$

曲线配对

# BLS m-of-n门限签名

- 示例：构建3个不同设备上的2-of-3多重签名方案（为1笔交易/消息共同签名）

➤ 已知信息 → ①设置阶段 → ②签名 → ③验签



P1 =  $pk_1 \times G$

a1 == hash(P1, {P1, P2, P3})  
a2 == hash(P2, {P1, P2, P3})  
a3 == hash(P3, {P1, P2, P3})  
  
 $P = a1 \times P1 + a2 \times P2 + a3 \times P3$

MK1 =  $(a1 \times pk_1 + a2 \times pk_2 + a3 \times pk_3) \times H(P, 1)$   
MK2 =  $(a1 \times pk_1 + a2 \times pk_2 + a3 \times pk_3) \times H(P, 2)$   
MK3 =  $(a1 \times pk_1 + a2 \times pk_2 + a3 \times pk_3) \times H(P, 3)$   
  
 $P = (a1 \times pk_1 + a2 \times pk_2 + a3 \times pk_3) \times G$

设置签名者编号

计算公钥

计算聚合公钥

为每个设备聚合签名  $S = pk \times H(m)$   
称为“成员密钥(Machine Key)”

每个成员密钥都是  
消息  $H(P_i)$  的有效  
n-of-n 签名

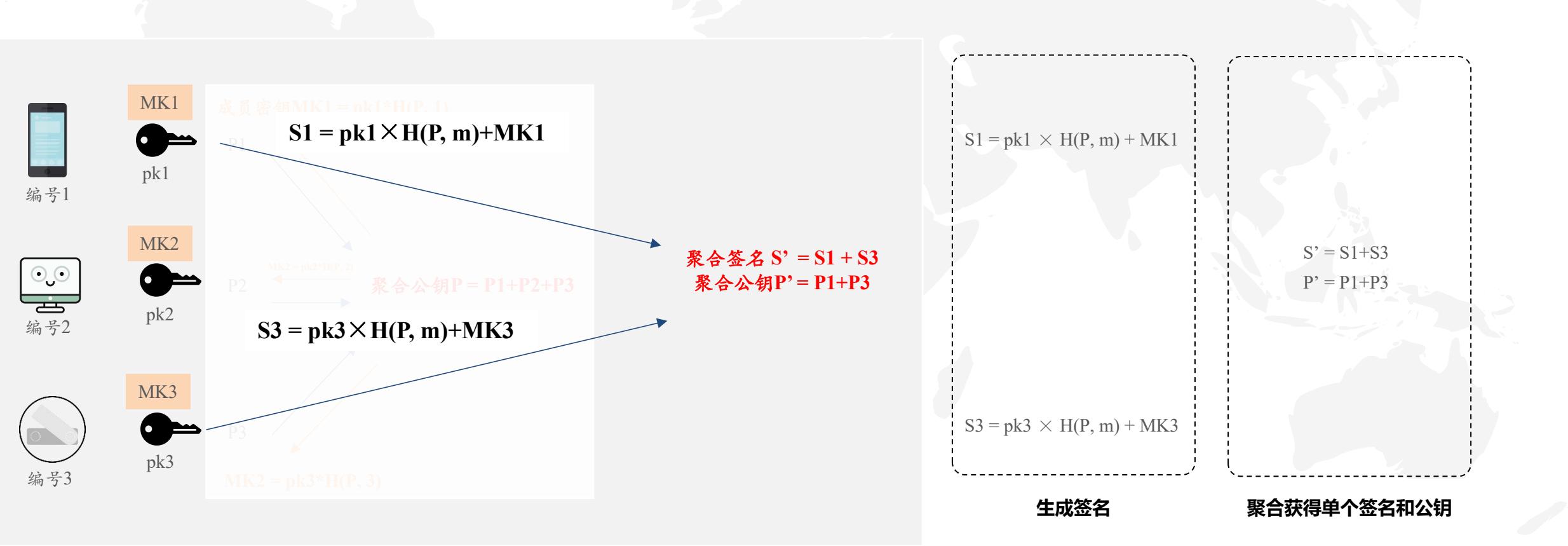
$$\begin{aligned} e(G, S) &== e(P, H(m)) \\ \downarrow \\ e(G, MK_i) &== e(P, H(P,i)) \end{aligned}$$

将用来证明这些设备是门  
限签名方案的有效参与者

# BLS m-of-n门限签名

- 示例：构建3个不同设备上的2 -of- 3多重签名方案（为1笔交易/消息共同签名）

➤ 已知信息 → ①设置阶段 → ②签名 → ③验签



# BLS m-of-n门限签名

- 示例：构建3个不同设备上的2-of-3多重签名方案（为1笔交易/消息共同签名）

➤ 已知信息 → ①设置阶段 → ②签名 → ③验签

$$e(G, S) == e(P, H(m))$$



$$e(G, S') = e(P', \underline{H(P, m)}) \times e(P, \underline{H(P, 1)+H(P, 3)})$$

# 提纲

一

多签、聚合与门限签名

二

Shamir秘密共享、多签与门限签名

三

BLS m-of-n门限签名

四

Gnosis Safe的签名方案 – 多签非门限

五

DVT技术与门限签名技术有何关联？

# Gnosis Safe的签名方案 – 多签非门限

- 多签钱包的特点：①允许参与方离线，②参与方签名公开

The screenshot shows a transaction history from Gnosis Safe. At the top, it indicates "Sent 0.0001 GOR to: gor:0x4f1c46445F2af322Dfb15512C69e2d8C2553B23d" at 1:58 PM. The transaction hash is Oxd623...3c26, and the safeTxHash is Ox5291...3100. It was created on 2022/7/19 13:54:02 and executed on 2022/7/19 13:58:15. The operation was a call to address 0. Advanced details show gas values and tokens sent to addresses gor:0x0000...0000 and gor:0x0000...0000. Two signatures were provided: "Signature 1" and "Signature 2", both 65 bytes long. A sidebar on the right shows the transaction status: "Created", "Confirmations (2 of 2)" (with two entries for the signers), "Hide all", and "Executed" (with one entry for the receiver). A blue arrow points down from the sidebar to the word "pending".

pending

# Gnosis Safe的签名方案 – 多签非门限

- 多签钱包的特点：①允许参与方离线，②参与方签名公开，③参与方其中之一丢了私钥，会导致交易签名无法完成

The screenshot shows a transaction details page from the Gnosis Safe web interface. At the top, it displays the number of signatures required (3), the current change threshold (changeThreshold), the time of creation (about 1 hour ago), the number of signatures received (1 out of 2), and the status (Awaiting confirmations).  
  
The main section shows a confirmation policy change (1 confirmation required) and the transaction details:

- safeTxHash: 0x761e...cfbe
- Created: 2023/3/26 14:26:21
- Advanced details:
  - Operation: 0 (call)
  - safeTxGas: 0
  - baseGas: 0
  - gasPrice: 0
  - gasToken: gor:0x0000...0000
  - refundReceiver: gor:0x0000...0000
  - Signature 1: 65 bytes
  - Raw data: 36 bytes

  
On the right side, there is a sidebar titled "Awaiting confirmations" which lists the current state of the transaction:

- Created
- Confirmations (1 of 2)
  - gor:0x4f1c...B23d
- Hide all
- Can be executed
  - Can be executed once the threshold is reached

At the bottom right of the sidebar are two buttons: "Confirm" and "Replace".

# Gnosis Safe的签名方案 – 多签非门限

## 多签钱包的特点：链上逐个验签

```
233 *
234 * @dev Checks whether the signature provided is valid for the provided data, hash. Will revert otherwise.
235 * @param dataHash Hash of the data (could be either a message hash or transaction hash)
236 * @param data That should be signed (this is passed to an external validator contract)
237 * @param signatures Signature data that should be verified. Can be ECDSA signature, contract signature (EIP-1271) or approved hash.
238 * @param requiredSignatures Amount of required valid signatures.
239 */
240 function checkNSignatures(
241     bytes32 dataHash,
242     bytes memory data,
243     bytes memory signatures,
244     uint256 requiredSignatures
245 ) public view {
246     // Check that the provided signature data is not too short
247     require(signatures.length >= requiredSignatures.mul(65), "GS020");
248     // There cannot be an owner with address 0;
249     address lastOwner = address(0);
250     address currentOwner;
251     uint8 v;
252     bytes32 r;
253     bytes32 s;
254     uint256 i;
255     for (i = 0; i < requiredSignatures; i++) {
256         (v, r, s) = signatureSplit(signatures, i);
257         if (v == 0) {
258             // If v is 0 then it is a contract signature
259             // when handling contract signatures the address of the contract is encoded into r
260             currentOwner = address(uint160(uint256(r)));
261
262             // Check that signature data pointer (s) is not pointing inside the static part of the signatures bytes
263             // This check is not completely accurate, since it is possible that more signatures than the threshold are send.
264             // Here we only check that the pointer is not pointing inside the part that is being processed
265             require(uint256(s) >= requiredSignatures.mul(65), "GS021");
266
267             // Check that signature data pointer (s) is in bounds (points to the Length of data -> 32 bytes)
268             require(uint256(s).add(32) <= signatures.length, "GS022");
269
270             // Check if the contract signature is in bounds: start of data is s + 32 and end is start + signature Length
271             uint256 contractSignatureLen;
272             // solhint-disable-next-line no-inline-assembly
273             assembly {
274                 contractSignatureLen := mload(add(add(signatures, s), 0x20))
275             }
276             require(uint256(s).add(32).add(contractSignatureLen) <= signatures.length, "GS023");
277
278             // Check signature
279             bytes memory contractSignature;
280             // solhint-disable-next-line no-inline-assembly
281             assembly {
282                 // The signature data for contract signatures is appended to the concatenated signatures and the offset is stored in s
283                 contractSignature := add(add(signatures, s), 0x20)
284             }
285             require(ISignatureValidator(currentOwner).isValidSignature(data, contractSignature) == EIP1271_MAGIC_VALUE, "GS024");
286         } else if (v == 1) {
287             //
```

# 提纲

一

多签、聚合与门限签名

二

Shamir秘密共享、多签与门限签名

三

BLS m-of-n门限签名

四

Gnosis Safe的签名方案 – 多签非门限

五

DVT技术与门限签名技术有何关联？

# DVT技术与门限签名技术有何关联?

## ➤ 以太坊2运行原理（极简理解）

- 多个节点组成了一个现实世界的计算机，每个节点保持同步，使以太坊网络保持着持续的联系，使EVM安全地保持同步
- 验证器是一个虚拟实体，由节点操作，存在于节点服务器中，参与PoS

不质押：只负责同步数据（视为轻节点）

质押：上链（提交区块） + 同步数据（视为全节点 / 矿工）

# DVT技术与门限签名技术有何关联?

## ➤ 以太坊2运行原理（极简理解）

- 多个节点组成了一个现实世界的计算机，每个节点保持同步，使以太坊网络保持着持续的联系，使EVM安全地保持同步
- 验证器是一个虚拟实体，由节点操作，存在于节点服务器中，参与PoS



# DVT技术与门限签名技术有何关联?

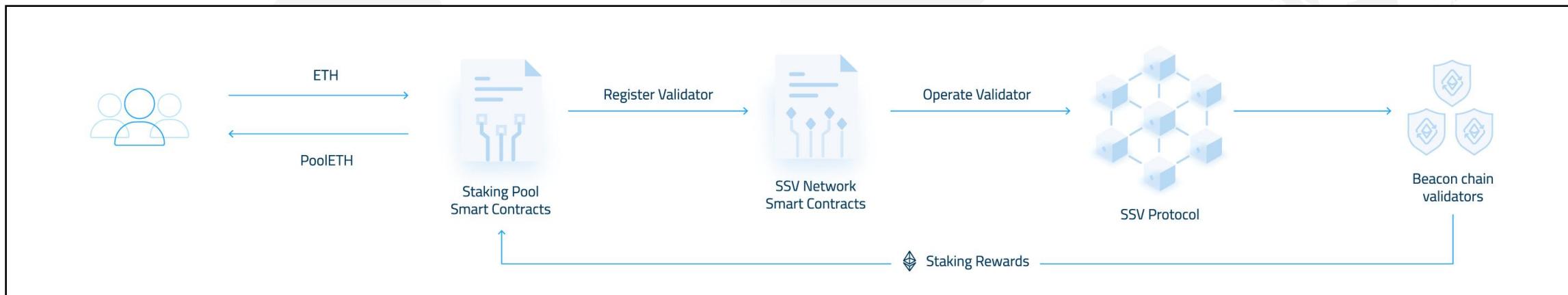
## ➤ 以太坊2运行原理（极简理解）

- 验证器是一个虚拟实体，由节点操作，存在于节点服务器中，参与PoS
- 将验证器考虑为以太坊节点（运行中的服务器）的验证密钥key
- DVT/SSV：将验证器key拆分为4个KeyShare的协议



# DVT技术与门限签名技术有何关联?

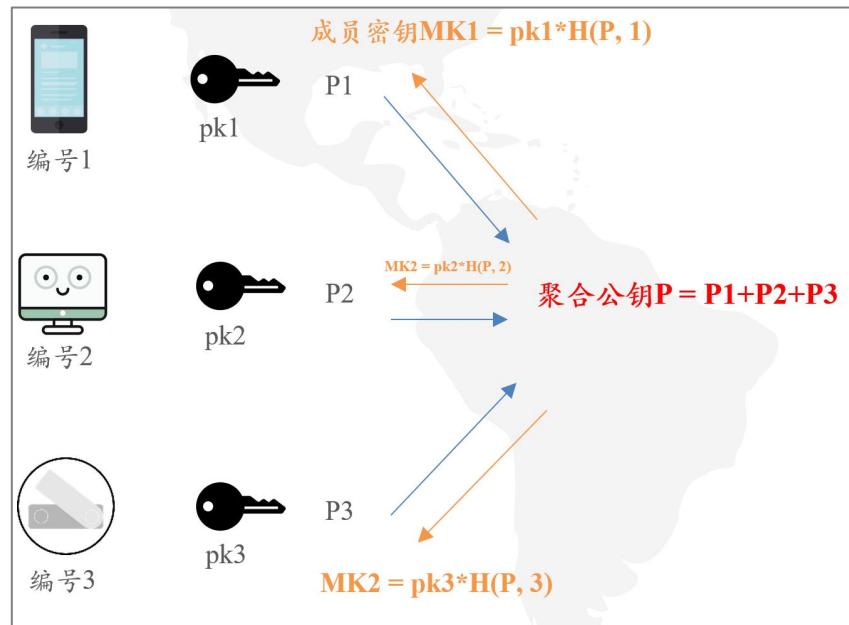
- DVT/SSV是以太坊2网络上一个可插拔的组件



# DVT技术与门限签名技术有何关联?

- DVT由4个关键部分组成：分布式密钥生成、Shamir 密钥共享、安全多方计算和DVT BFT共识层

- 将运营商节点与前面提及的单个设备，如手机、电脑、硬件钱包



• MKi为每个运营商节点的KeyShare

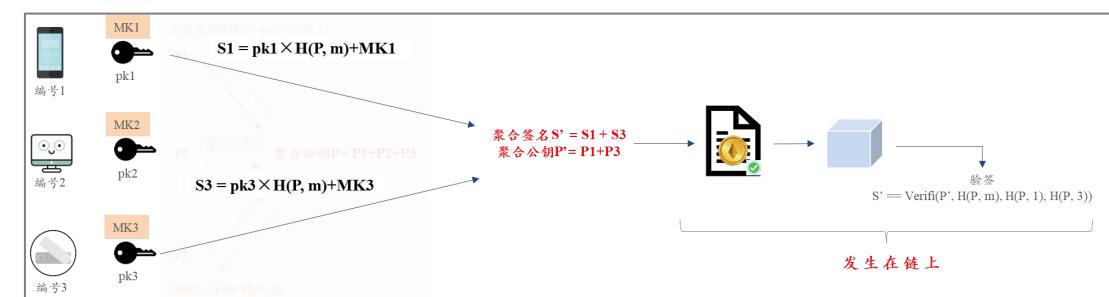
• 每一个Mki的生成过程都是一次Shamir 密钥共享的过程

• ssv官网上提到多方安全计算应用于秘密共享，个人理解从密钥生成到聚合签名都属于多方安全计算（事实上门限签名就是多方安全计算的一个分支）

## 分布式密钥生成

## Shamir 密钥共享

## 多方安全计算



- 个人理解：DVT技术就是本质上就是对门限签名算法的应用

- 尚未明确：分布式密钥生成是否为DKG技术，还是说单纯的依赖可信第三方（dealer）的Shamir 密钥共享？

# 尚未解答的系列问题

---

- ✓ BLS、SSS、门限、多签四个关键词之间的关系是什么？
- ✓ 多方安全计算与门限签名的关系是什么？
- TSS 与 DVT、DPKI 的关系是什么？能不能说TSS是他们的底层基础设施呢？
- ✓ BLS 签名与ECDSA 签名的关系？
- 相关产品：ZenGo,其初衷是不是要解决私钥安全问题呢？
- ✓ Gnosis safe使用的是多重签名吗？
- Ronin多签安全问题：Axie DAO Validator将他们的多重签名借给了 Sky Mavis  
→ 怎么借的？是将私钥借出了吗？
- LAZARUS GROUP APT组织是不是正在往私钥盗窃的方向发展？
- 基于MPC的门限签名:BitiZen钱包门限签名方案？ZK可信设置？DVT？
- 签名安全本质上是否完全等价于私钥安全？