

MPC 多方计算钱包

- [参考资料](#)
- [背景](#)
 - [钱包分类](#)
 - [加解密相关](#)
 - [对称加密](#)
 - [非对称加密](#)
 - [ECSDA椭圆曲线算法（ECC与DSA的结合）](#)
 - [Schnorr算法（可用于签名聚合、密钥聚合）](#)
 - [BLS签名算法（签名聚合、密钥聚合）](#)
 - [Paillier加密方案（属于同态加密中的加法同态）](#)
 - [迪菲-赫尔曼加密算法](#)
 - [Shamir's Secret Sharing\(SSS, Shamir密钥分享\)](#)
 - [Verifiable Secret Sharing\(VSS, 可验证的密钥分享\)](#)
 - [混合加密](#)
 - [Multiplicative-to-Additive \(MtA\) 协议](#)
 - [TSS阈值签名相关算法](#)
 - [1. ECDSA阈值签名算法: GG18](#)
 - [2. ECDSA 门限签名算法: GG20](#)
 - [3. DMZ21](#)
 - [总结及开源代码汇总](#)
 - [椭圆曲线相关运算](#)
 - [数学: 多项式函数](#)
- [MPC \(SMPC\)](#)
 - [私钥分片](#)
 - [MPC-- 阈值签名方案\(TSS\): 分布式密钥生成+ 分布式签名+阈值签名](#)
 - [基于RSA的阈值签名算法流程](#)
 - [阈值签名与区块链相结合](#)
 - [门限签名的算法原理与落地实践](#)
 - [MPC钱包](#)
 - [核心流程](#)
 - [MPC钱包的优点](#)
















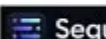





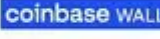





- [MPC钱包的缺点](#)
- [各签名方式：多重签名 VS Secret Sharing VS 阈值签名](#)
- [MPC钱包 vs 多签钱包](#)
- [账户抽象（Account Abstract） vs MPC](#)
- [MPC钱包竞品分析](#)
- [Safeheron](#)
 - [ZenGo](#)
 - [ZenGo注册流程](#)
 - [Fireblocks](#)
 - [Lit](#)
 - [Qredo](#)
 - [coinbase wallet](#)
- [my](#)

参考资料

- <https://foresightnews.pro/article/detail/19288>

背景

钱包分类

Conventional Wallets	Smart Contract Wallets	MPC Wallets
 METAMASK  imToken  Rabby Wallet  Frame  Minerva  Trust Wallet Hardware Wallets  Ledger  TREZOR  keepkey  BitBox	 Safe  argent  AMBIRE  Pillar  linen  Sequence  UNIPASS	 Lit  Qredo  ZenGo  Fireblocks  coinbase WALLET  SEPIOR  bitpowr  SAFEHERON  entropy  OpenBlock

如上图所示，目前钱包主要分为：传统钱包、智能合约钱包、MPC钱包 三大类。

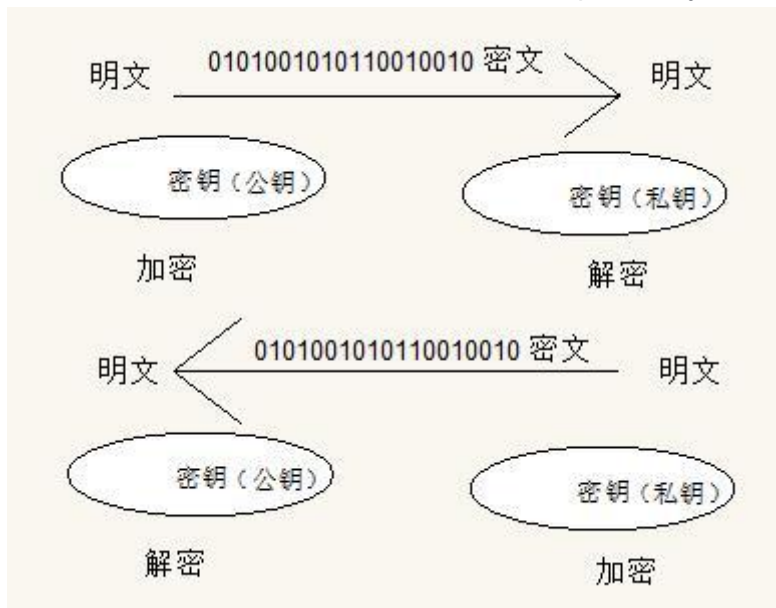
加解密相关

对称加密

- 加密和解密时使用相同的密钥
- 对称加密存在对称加密密钥被劫持的问题，但是速度较快

非对称加密

- 非对称加密算法需要两个密钥：公开密钥（publickey）和私有密钥（privatekey）



- 如果用公钥对数据进行加密，只有用对应的私钥才能解密；如果用私钥加密，只有对应的公钥才能验证
- 非对称加密算法实现机密信息交换的基本过程是：甲方生成一对密钥并将其中的一把作为公用密钥向其它方公开；得到该公用密钥的乙方使用该密钥对机密信息进行加密后再发送给甲方；甲方再用自己保存的另一把专用密钥对加密后的信息进行解密。甲方想要回复乙方时正好相反，使用乙方的公钥对数据进行加密，同理，乙方使用自己的私钥来进行解密。另一方面，甲方可以使用自己的私钥对机密信息进行签名后再发送给乙方；甲方再用乙方的公钥对乙方发送回来的数据进行验签。
- 当今使用的最常见的非对称加密算法之一称为 RSA。在 RSA 方案中，密钥是使用模数生成的，该模数是通过将两个数字（通常是两个大质数）相乘得出的。简而言之，模数生成两个密钥（一个可以共享的公钥，一个应该保密的私钥）。RSA 算法于 1977 年由 Rivest、Shamir 和 Adleman（因此称为 RSA）首次描述，至今仍是公钥密码系统的主要组成部分。
- 非对称加密存在的问题：加密速度慢，不适合用来传输少量数据，浪费性能；从前面的讲解来看，非对称加密是一种比较安全的方式，安全的地方在于没有对应的钥匙几乎不可能破解加密后的数据，但是，在公钥的交付过程中可能会被攻击人拦截到并且替换成自己的公钥，让甲方和乙方在自以为很安

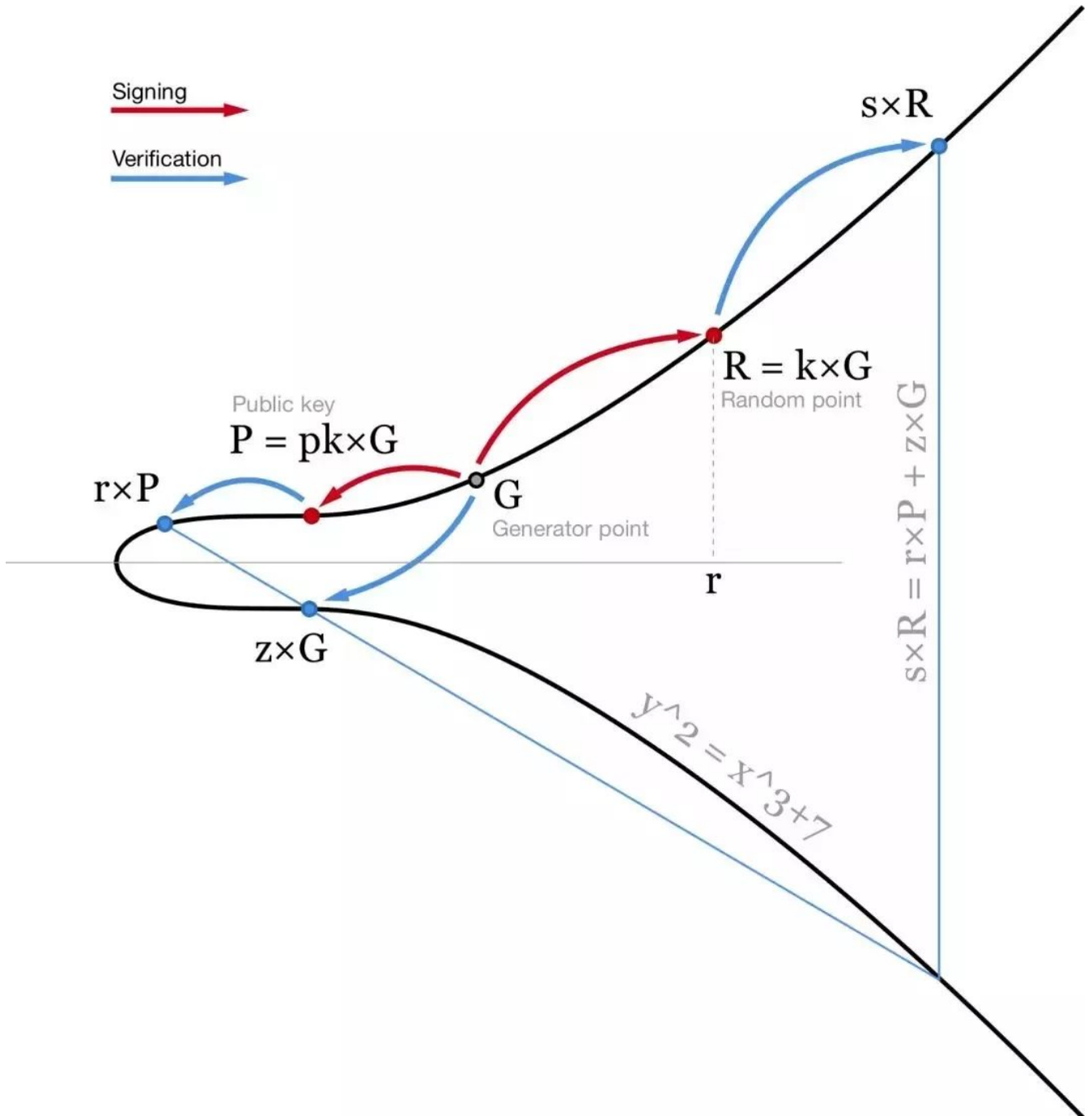
The diagram illustrates a man-in-the-middle attack on a public key infrastructure. It is divided into three sections: A (Sender), B (Receiver), and a central '坏人' (Bad Man) section.

- Section A (Sender):** A box labeled 'abc' sends a message to a red box labeled '坏公钥' (Bad Public Key). This box then sends a garbled message '&*@#¥¥#' to a red box labeled '坏私钥' (Bad Private Key).
- Section B (Receiver):** A red box labeled 'xyz' sends a message to a white box labeled '私钥' (Private Key). This box then sends the message to a white box labeled '公钥' (Public Key).
- Central Section (坏人 - Bad Man):**
 - The '坏私钥' (Bad Private Key) sends the garbled message '&*@#¥¥#' to a white box labeled 'abc'.
 - The '坏公钥' (Bad Public Key) sends a message to a white box labeled 'B的公钥' (B's Public Key).
 - The '坏人' (Bad Man) intercepts the message from 'xyz' to 'B的公钥' and modifies it (篡改) to a garbled message '***&*#¥#', which is then sent to the '私钥' (Private Key).

The '坏人' (Bad Man) section is highlighted with a red border and contains the text '坏人' in red.

ECSDA椭圆曲线算法 (ECC与DSA的结合)

- 属于非对称加密

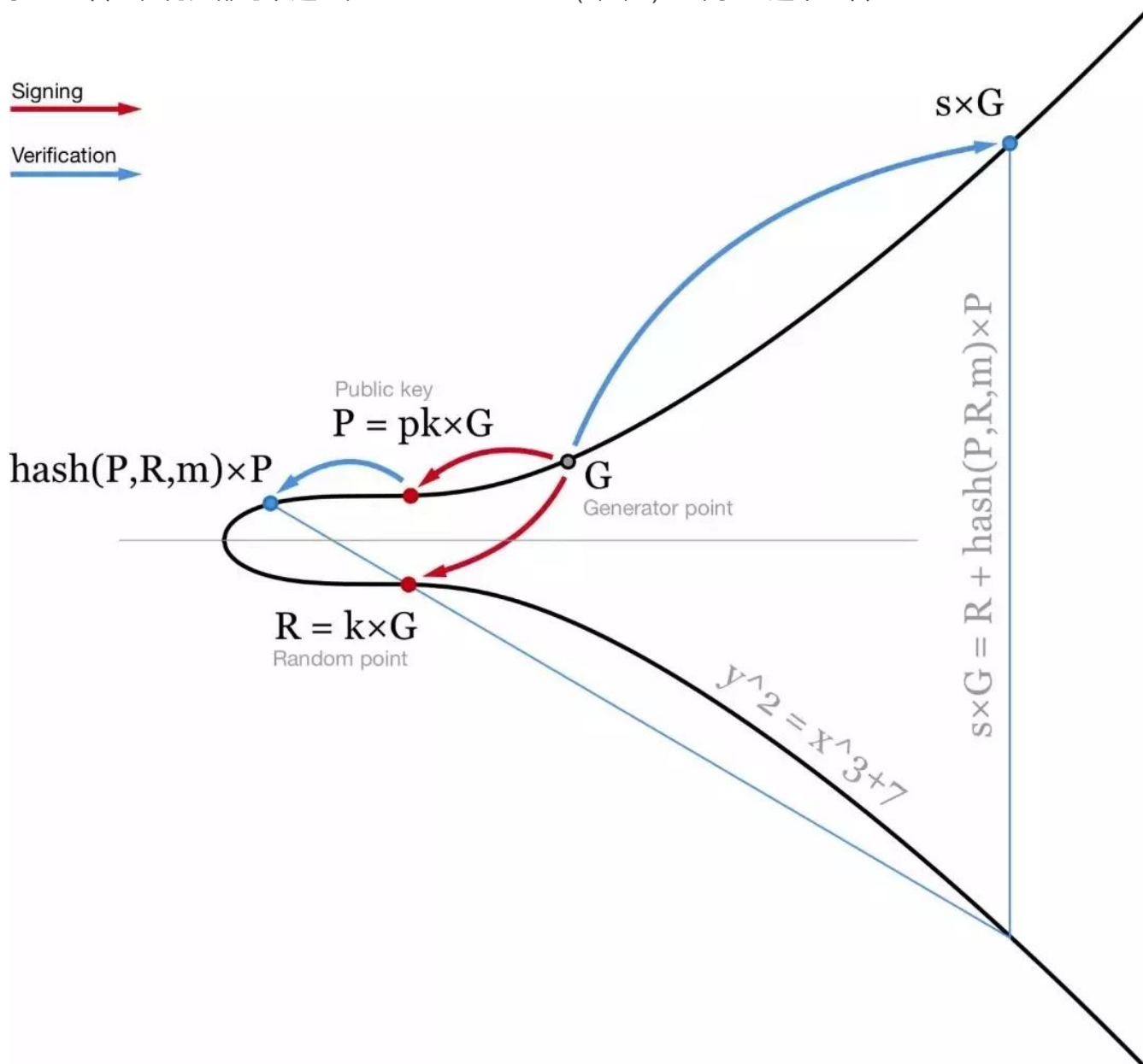


- 私钥 pk , 公钥为 $P=pk \times G$
- 消息 m 进行签名：首先对消息进行哈希处理, $z=\text{hash}(m)$; 并找一个随机数字 k , m 对应的签名为随机点 R 的 x 坐标(kG)和 $s=(z+rp_k)/k$
- 验证签名：任何人都可通过检查点 $(z/s) \times G + (r/s) \times P$ 的 x 坐标等于 r , 来验证我们的签名

Schnorr算法 (可用于签名聚合、密钥聚合)

- 属于非对称加密, 与ECSDA相比略有不同
- 消息 m 进行签名：随机点 R 的 x 坐标($k \times G$) 和 $s = k + \text{hash}(P, R, m) \cdot pk$

- 验证签名：任何人都可以通过检查 $s \times G = R + \text{hash}(P, R, m) \times P$ 来验证这个签名



3. Schnorr签名的批量验证

要验证比特币区块链中的区块，我们需确保区块中的所有签名都有效。如果其中一个是无效的，我们不会在乎是哪个无效的，我们只会拒绝掉整个区块。

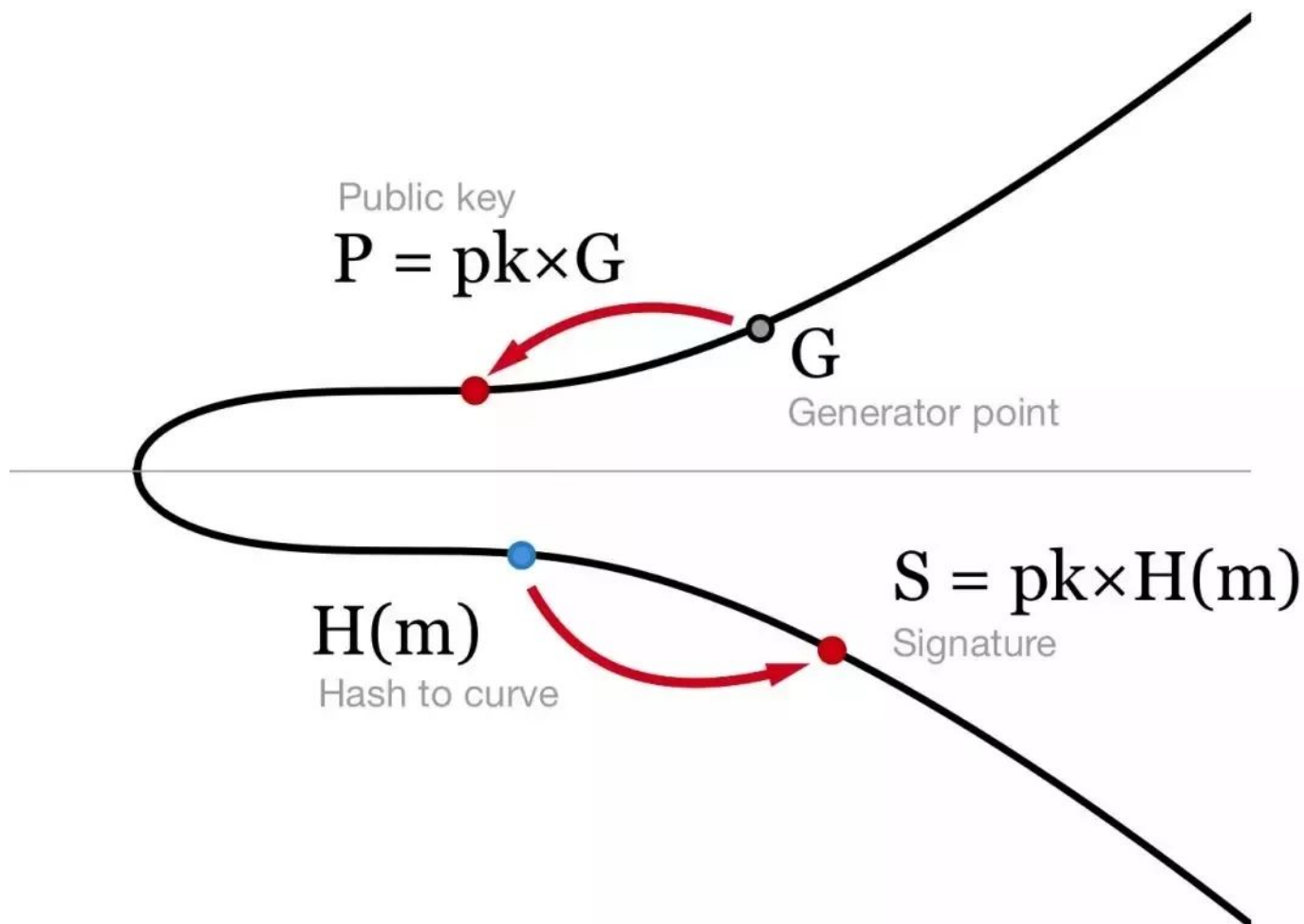
对于ECDSA签名算法，每个签名都必须单独验证。也就是说，如果区块中有1000个签名，我们就需要计算1000次倒置和2000次点乘运算，总共约3000次繁重的计算任务。

而通过使用Schnorr签名，我们可以将所有签名验证方程相加，从而节省一些计算能力。对于有1000个签名的区块而言，我们需验证：

$$(s_1 + s_2 + \dots + s_{1000}) \times G = (R_1 + \dots + R_{1000}) + (\text{hash}(P_1, R_1, m_1) \times P_1 + \text{hash}(P_2, R_2, m_2) \times P_2 + \dots + \text{hash}(P_{1000}, R_{1000}, m_{1000}) \times P_{1000})$$

BLS签名算法（签名聚合、密钥聚合）

- BLS签名可以修复Schnorr中存在的一些问题：不需要随机数，区块中的所有签名都可以组合成单个签名，m-of-n类型的多重签名非常简单，我们不需要签名者之间进行多轮通信。此外，BLS签名相比Schnorr签名或ECDSA签名要小2倍，其签名不是一对，而是一个单曲线点。
- 消息m进行签名：将消息哈希到曲线 $H(m)$ ，并将结果点乘以私钥： $S = pk \times H(m)$ 即为签名
- 验证签名：检查 $e(P, H(m)) = e(G, S)$ ，（BLS签名验证中，只需要检查公钥和消息哈希是否映射到与曲线生成器点和签名相同的数字）



- [理解 BLS 签名算法](#)

Paillier加密方案（属于同态加密中的加法同态）

$$[x] := g^x r^n$$

$$[x] \cdot [y] = g^{x+y} (r_x r_y)^n$$

迪菲- 赫尔曼加密算法

- 该算法针对性地解决了对称加密中密钥的安全性问题。
- 可参考：[迪菲-赫尔曼密钥交换](#)

Shamir's Secret Sharing(SSS, Shamir密钥分享)

原始的SSS，使用m of n 阈值签名，会存在以下问题：

- 密钥分发者知晓完整的密钥，有作恶的可能，例如对部分秘密持有者发放错误的分片数据。
- 密钥分片的持有者可能提供非真实的分片数据

Verifiable Secret Sharing(VSS, 可验证的密钥分享)

- VSS概念由Benny Chor, Shafi Goldwasser, Silvio Micali等人在1985年首次提出，是为了解决Shamir密钥分享方法存在的问题的。
- 此处直接讲实用的Feldman的VSS方案，它是在SSS的基础上进行了一些修改（多了mod q相关逻辑）

Feldman方案

实际工程中使用的密钥分享都是有限域上的循环群的运算，采用公共g作为生成元。

为了使得分发的秘密碎片的数据可验证，秘密分发者除了给出秘密的分片数据外，还要提供对应的系数承诺（ c_0, c_1, \dots ）。

符号约定

设 p 是一个大素数， q 为 $p-1$ 的一个大素数因子， g 属于 Z_p^* 且为 q 阶元素， (p, q, g) 是公开可知的， n 是参与者的数目， s 为要共享的密钥， k 是门限值。

秘密分发阶段

选定多项式：

$$a(x) = a_0 + a_1x + a_2x^2 + \dots + a_{k-1}x^{k-1}$$

a_0 代表密钥，计算承诺（commitment）：

$$c_0 = g^{a_0}, c_1 = g^{a_1}, \dots, c_{k-1} = g^{a_{k-1}}$$

以上运算是在mod p基础上的。

将承诺 c_i 和 s_i [注：如同上篇是若干选定的点]一同发送给参与者 p_i 。

【具体工程实现中，由于承诺对所以参与者都是一样的，也可先广播承诺给所有参与者，然后单独秘密发送分片，以减少消息传输量】。

其次，当第 i 个参与者，收到数据碎片 v 时，作如下验证：

$$g^{s_i} = \prod_{j=0}^{k-1} (c_j)^{v_j} \bmod p$$

容易验证等式成立。

由于承诺绑定了系数，如果分发者给出承诺不是用多项式方程真实系数，会导致验证失败。

秘密重构阶段

当一个参与者提供他保存的分片数据（ c_i 和 s_i ）时，其他参与者会做同样的验证，这样可以保证参与者在恢复阶段的诚实行为。

Feldman的方案的安全性是建立在离散对数问题的困难性基础上的，所以它是条件安全的。

到此，我们提到过多次安全性，下面我看看密码学中的安全性如何定义的？

混合加密

- 对称加密和非对称加密相结合。首先，把对称加密的密钥通过非对称加密的方式发送给对方，防止对称加密的密钥半路被拦截，接下来的通信中使用的还是对称加密方式。
- 混合加密 其实是没有解决 非对称加密存在的“中间人攻击”问题的。需要再添加数字签名来解决。
- 可参考：[混合加密解析](#)

Multiplicative-to-Additive (MtA) 协议

假设 Alice 有个秘密值 $a \in \mathbb{Z}_q$ ，而 Bob 有个秘密值 $b \in \mathbb{Z}_q$ ，下面介绍一个协议可以在 Alice 不泄露 a ，Bob 不泄露 b 的情况下，Alice 和 Bob 分别得到另外一个秘密值 $\alpha \in \mathbb{Z}_q, \beta \in \mathbb{Z}_q$ ，满足：

$$ab = \alpha + \beta \bmod q$$

这个协议称为乘法到加法的转换协议（MtA）。

图 4 是 MtA 协议的示意图，协议中用到了 Paillier 同态加密。

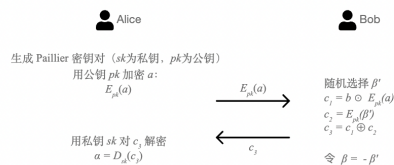


Figure 4: Multiplicative-to-Additive (MtA) 协议的示意图

Alice 把 a 使用 Alice 公钥进行加密后，给 Bob；而 Bob 利用 Paillier 同态加密性质，把加密（当然也是 Alice 公钥加密）后的 $ab - \beta$ 给 Alice。这样 Alice 使用私钥解密后得到的值记为 α ，它们会满足 $ab = \alpha + \beta \bmod q$ （注：这里不是很严谨，Paillier 解密后得到的值是 $ab - \beta \bmod N$ ，而 MtA 协议要求的是对 q 取模，下一节会介绍当 $ab - \beta < N$ 时，对 q 取模也一样成立）。

- 参考：<http://aandds.com/blog/multiparty-threshold-ecdsa.html#dcef8ba4>

TSS阈值签名相关算法

整个过程分为两步：

1. Key generation：分布式协作生成各个私钥碎片
2. Sign：签名时会用到VSS、MtA（Multiplicative-to-Additive，乘法到加法的转换协议，用到了Paillier 同态加密）等

1. ECDSA阈值签名算法：GG18

- 参考：<http://aandds.com/blog/multiparty-threshold-ecdsa.html#8a760809>
- 2018 年 Rosario Gennaro 和 Steven Goldfeder 在论文 Fast Multiparty Threshold ECDSA with Fast Trustless Setup 中提出的方案，简称 GG18 方案。
- 在GG18方案中，存在一个中心化的dealer，dealer有可能会获取到完整的私钥
- GG18 协议，如果有恶意参与者，那么协议会终止，但在某些场景下却不知道谁是恶意参与者（在 GG20 协议中，只要默认失败，就可识别出是谁的责任）

2. ECDSA 门限签名算法：GG20

- Rosario Gennaro 和 Steven Goldfeder 于 2020 年发表论文 One Round Threshold ECDSA with Identifiable Abort，被称为 GG20。
- GG20的主要特点：Dealerless(取而代之的是，交互消息时互为dealer)；Rounds更少；可识别恶意参与者
- GG18 的 sign 过程需要 9 rounds，而 GG20 的 sign 过程只需要 7 rounds（前 6 个 rounds 和待签名的 msg 无关，可提前进行；最后一个 round 才和 msg 相关）。
- GG20使用了下列密码学或数学手段：分布式密钥生成、Paillier同态加密、Shamir秘密共享、Feldman VSS、MtA转换、零知识证明、拉格朗日插值、不可延展二义性承诺

3. DMZ21

- 比GG20更快、轮次更少

总结及开源代码汇总

MPC钱包较多、但目前相关开源内容较少，

1. 币安开源的是GG18
- 2.

椭圆曲线相关运算

- <https://www.cnblogs.com/Duxue/p/15008735.html>

数学：多项式函数、

- 多项式函数：形如 $f(x)=a_n \cdot x^n + a_{n-1} \cdot x^{(n-1)} + \dots + a_2 \cdot x^2 + a_1 \cdot x + a_0$ 的函数（当 $n=1$ 是，为一次函数； $n=2$ 时，为二次函数）
- N元一次方程式：
- 拉格朗日（插值）多项式：

MPC (SMPC)

- MPC 又叫 SMPC，全称为 Security Multi-Party Computation（安全多方计算），是密码学的一个分支。多方计算是安全可信计算中的一个分支，安全可信计算主要有两大类：外包计算和多方计算。
- MPC也可用作隐私计算
- 安全多方计算于1986年由姚期智院士通过**姚氏百万富翁问题**提出：两个百万富翁街头邂逅，他们都想炫一下富，比一比谁更有钱，但是出于隐私，都不想让对方知道自己到底拥有多少财富，如何在借助第三方的情况下，让他们知道谁更有钱。姚氏“百万富翁问题”后经发展，成为现代密码学中非常活跃的研究领域，即安全多方计算。
- MPC的目的：允许多个持有各自私有数据的参与方，共同执行一个计算逻辑、得到计算结果（在不同的学科中，而不仅仅是加密）。但整个过程中，参与的每一方均不会泄露各自数据。
- 在密码学中，MPC对于保管用于解密数据或生成数字签名的私钥特别有用。
- MPC 的两个主要属性是正确性和隐私性：
 - 正确性：算法产生的输出是正确的（如预期的那样）。
 - 隐私：一方持有的秘密输入数据不会泄露给另一方。
- 举个例子，假设某公司的 n 名员工想知道他们中的哪些人收入更高而没有透露其实际工资。在这里，敏感的输入是薪水，输出是薪水最高的员工的姓名。使用MPC的整个过程中不会透露任一员工的工资。

私钥分片

- 广义的私钥分片指的是将完整的私钥分割成多份，每一份为一个私钥分片，这样可以在一定程度上解决因为一个私钥丢失或被盗导致资产丢失的问题。
- MPC中的私钥分片指的是，在生成钱包前通过 MPC 算法生成多份密钥分片；交易时，由多份密钥分片通过 MPC 计算出签名。

- 与广义的私钥分片不同的是，通过 MPC 算法生成多份密钥分片，在整个过程中没有合并成过原始的完整私钥，从而实现高度安全。

MPC-- 阈值签名方案 (TSS) : 分布式密钥生成+ 分布式签名+阈值签名

Web3领域，MPC 通常用于**将一个私钥打碎成多片**，每个碎片分配给不同的各方来协作签署交易，而不泄露完整的私钥。

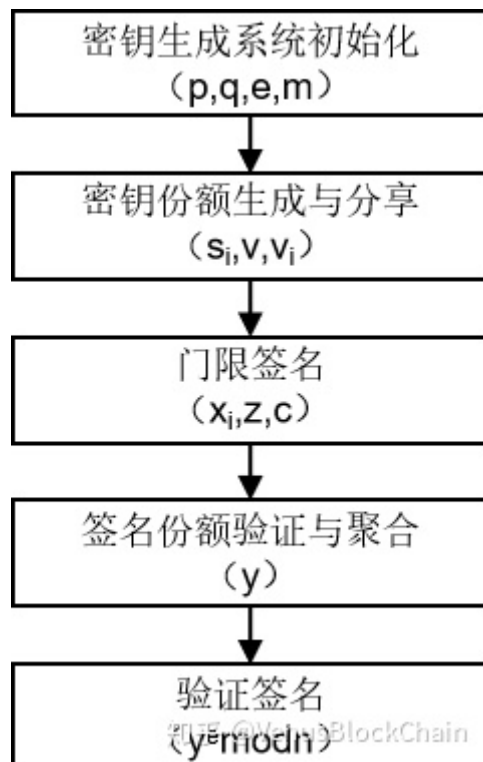
通过 MPC 可以做到从生成、使用、存储整个生命周期中，私钥可用、但不可见。

MPC包含很多技术方案，在目前 Web3 的语境下大都指的是 tss。

- TSS: Threshold Signature Scheme, 阈值（门限）签名方案，是一种MPC 方案。密钥碎片以分布式方式各自生成，而无需在任何一台机器上生成整个密钥、然后分片、最后分发出去。
- TSS允许灵活的定义阈值策略。例如，三个用户共同管理一个私钥。每个人都只持有部分私钥碎片，为了进行交易签名，需要将至少两个用户轮流进行签名，然后将签名数据整合在一起才能构建出最终有效的签名。
- 一个 m - n 的 tss 指的是一个公钥对应了 n 个私钥碎片，其中 m 个碎片的联合签名可以被公钥验签成功。不难发现这个逻辑类似于多签（multi-sig），他们的区别主要在公钥的数量上。
- 对于个人而言，阈值签名机制可以帮助使用持有的多个设备共同管理私钥，从而使单个受损设备不会对资产造成风险。对于业务运营商，阈值签名机制可实现更好的访问控制策略，以防止内部或外部人员窃取公司资金。

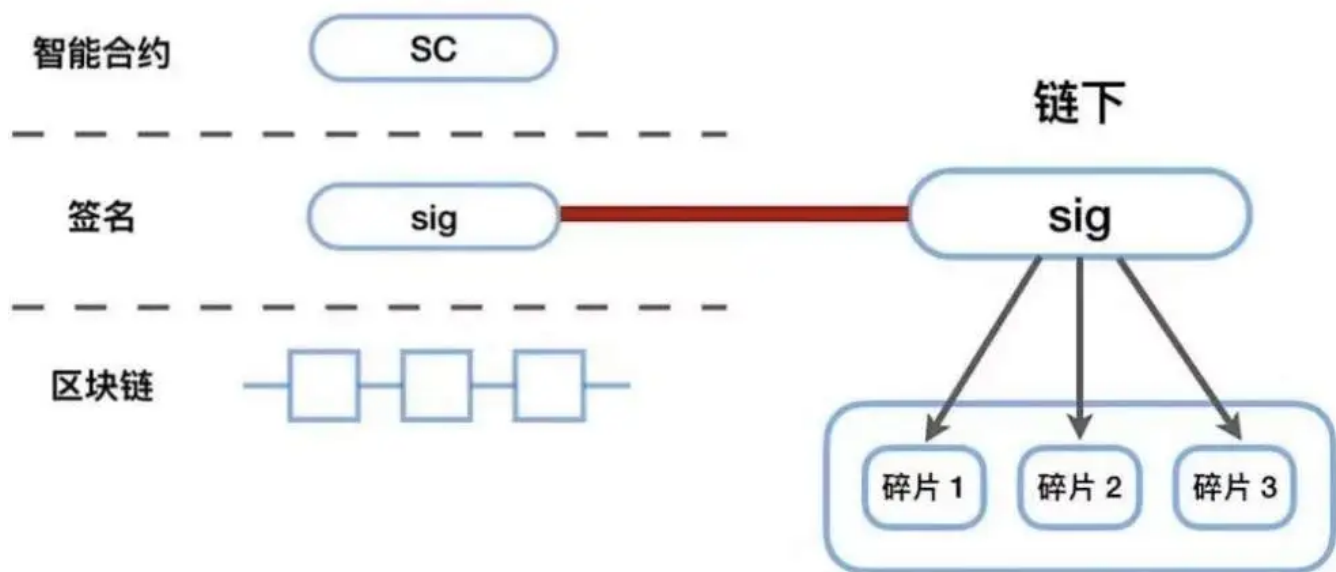
基于RSA的阈值签名算法流程

- 算法流程：



- 具体推导流程可参考：<https://zhuanlan.zhihu.com/p/135258941>

阈值签名与区块链相结合



门限签名的优势在于签名的生成是通过链下的 MPC 协议产生的，其结果是更加安全，避免了合约被黑客攻击的风险。因为门限签名与合约模块是**完全解耦**的，合约不需要理解签名的协议，它只要确认签名的有效性，这与传统的合约验签模式完全一致的。此外，合约的设计策略可以更加灵活，因为**除了验签外的大部分流程都搬到了链下**，使用方可以根据场景制定自己的碎片管理策略。

门限签名的算法原理与落地实践

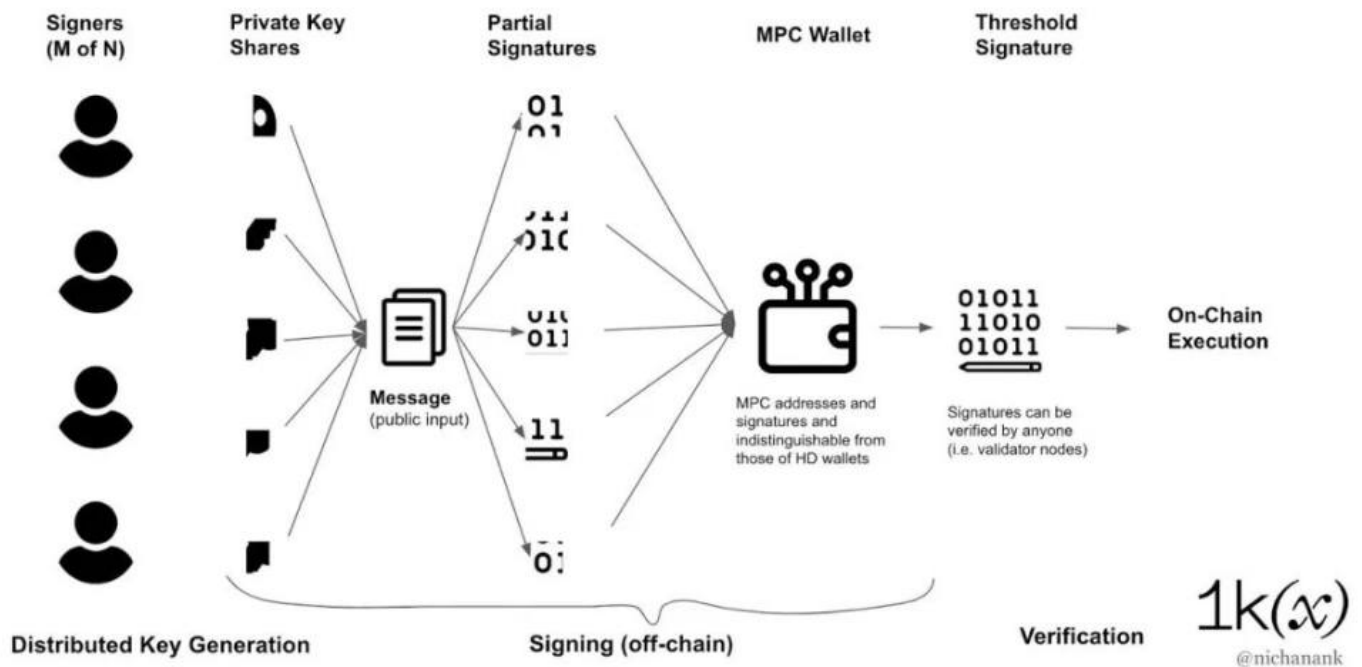
- Open TSS：开源，代码采用安全高效的 Rust 实现，支持一站式 ECDSA MPC 密钥生成(Keygen)、MPC 签名(Sign)。
- 项目地址：<https://github.com/LatticeX-Foundation/opentss>

MPC钱包

MPC 钱包通过使用阈值签名方案 (TSS) 消除了单点故障。在这个范式下，我们分布式创建私钥的各部分，这样就没有一个人或机器能够完全控制私钥——这个过程被称为分布式密钥生成 (DKG)。然后，我

们可以通过合并部分，并且在不暴露各方之间的部分的情况下共同生成公钥。

Multi-Party Computation (MPC) Wallets



为了对消息和交易进行签名，每一方都要输入“Private Key Shares”与公共输入（要签名的消息），以生成数字签名。任何知道公钥的人（即验证者节点）都能够验证签名。由于密钥部分是被组合的，**签名是在链下生成的，因此从 MPC 钱包生成的交易与传统的私钥钱包的交易没有区别（如上图所示，链上是感知不到的）。**

核心流程

1. 密钥生成：第一步也是最复杂的。我们需要生成一个公开的密钥（由公钥就可以确定EOA地址），用于验证未来的签名。但是，我们还需要为每一方生成一个单独的秘密，称为private key share。在正确性和隐私方面，我们说**该函数将向所有各方输出相同的公钥，并为每一方输出不同的秘密共享**，这样：（1）隐私：各方之间不会泄露秘密共享数据，以及（2）正确性：公钥是秘密份额的函数。
2. 签署交易时：不是单方使用他们的私钥签名，而是在多方之间运行**分布式签名**生成。因此，只要有足够多的人诚实行事，每一方都可以产生有效的签名。我们再次从本地计算（单点故障）转移到交互式计算。分布式密钥生成可以以允许不同类型访问结构的方式完成：一般的“t out of n”设置将能够承受私钥相关操作中最多 t 次任意失败，而无需危及安全。

MPC钱包的优点

- **无单点故障。**一个完整的私钥在任何时候都不会集中在一台设备上。也没有助记词。
- **可调整的签名方案。**授权的法定人数可以随着个人和组织需求的变化而变化，同时不改变地址。组织可以动态调整签名方案，而不必每次都通知交易对手一个新的地址。
- **细粒度访问控制。**机构用户可以为一个策略分配无限数量的交易审批者，并委派能够准确反映组织角色和安全措施（时间锁、MFA多因素验证、欺诈监控）的权限。个人可以通过MPC钱包即服务（wallet-as-a-service）选择半托管方式，由第三方持有其中部分密钥分片。
- **更低的交易成本和密钥恢复成本。**MPC钱包在区块链上表示为单个地址，其gas费与常规私钥地址相同。这对于每天进行数百笔交易的用户（例如在B2C用例中）来说非常重要。丢失的密钥分片也可以

在链下恢复。

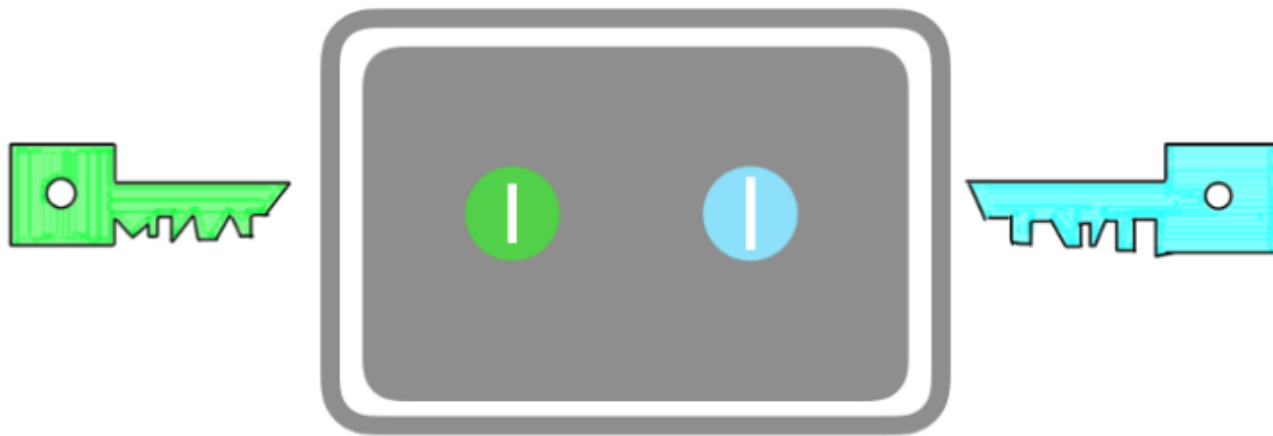
- **区块链无关性。** 密钥生成和签名依赖于链下纯密码学。与新的区块链兼容很容易，因为钱包只需要能够使用该链可识别的算法生成签名就可以了。

MPC钱包的缺点

- 链下问责制。签名授权策略和授权法定人数是在链下管理的，因此这些自定义规则仍然容易出现中心化故障。密钥分片仍然是加密秘密，应该拥有与完整私钥相同的处理方式。链下规则和签名阻碍了透明度，需要更严格的运营审计。
- 与多数用户采用的的大多数传统钱包不兼容（没有助记词，没有完整的私钥存储在单个设备上）。MPC算法并没有标准化，也没有得到机构级安全设备（如iPhone SEP和HSMs）的原生支持。
- 大多是单独定制产品。许多MPC库和解决方案都不是开源的，因此，如果出现问题，生态系统很难对它们进行独立审计和集成，很难进行事故分析。
- 如今，基于MPC的解决方案主要面向基金、家庭办公室、交易所和托管人等机构客户。Fireblocks和Qredo等MPC技术提供商使客户能够为不同类型的交易定义自己的工作流，并且能够让他们保持合规和安全。然而，散户投资者仍然依赖独立研究和私人密钥钱包。Web3Auth最近发布了一个MPC SDK，允许任何钱包或dapp借此成为用户的“web3原生多因素验证”，让用户可以使用自己的iCloud或电子邮件进行备份。去中心化托管协议（如Entropy）正在为消费者和DAO开发开源工具，以在线存储资产，并通过MPC设计交易的安全预防措施。

各签名方式：多重签名 VS Secret Sharing VS 阈值签名

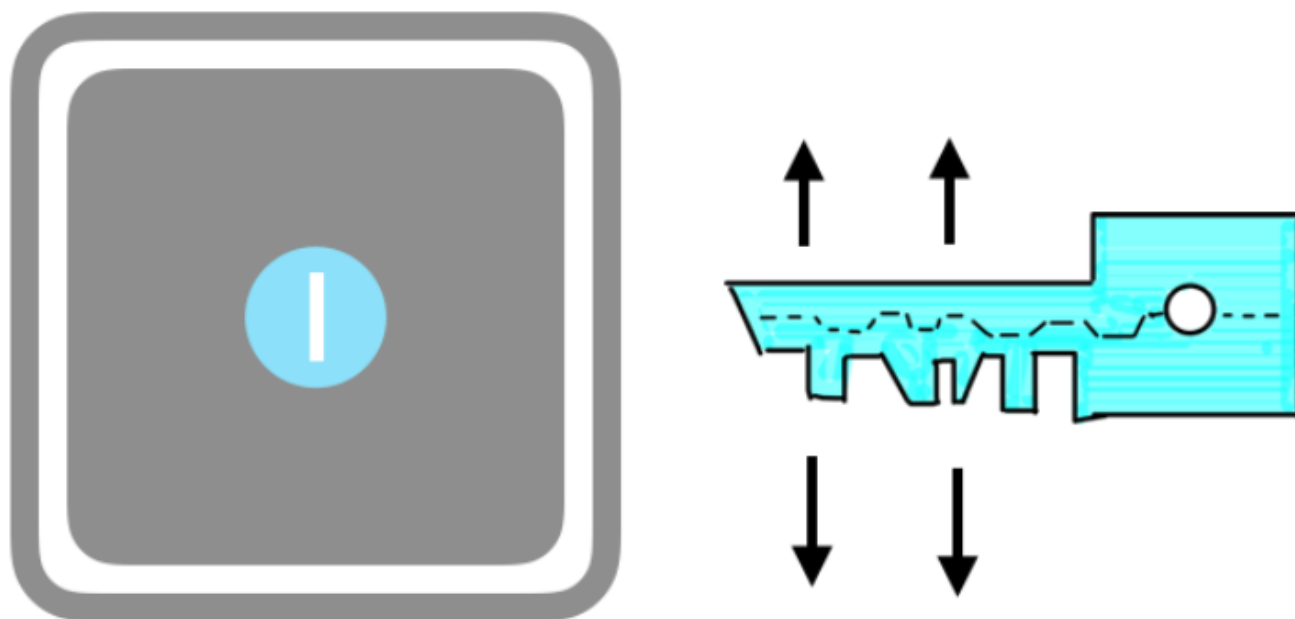
1. 多签签名：通常需要有多多个私钥（N），只有当其中大于等于M个私钥都签名授权后，才能执行交易
- 可以理解为：一个有多把钥匙的保险柜，需要多把钥匙才能打开



MultiSig Vault: Two Keys, Two Locks. The vault is “heavier” and looks different

2. 密钥共享（Secret Sharing, SS）模式：将同一个密钥分成多个部分（称作 key pieces）并以冗余方式分开保管。
- 优点：不使用时，key pieces可以保存在不同的地方
 - 缺点：初次创建、或者要使用该完整密钥时，密钥会以完整形式展示，此时攻击者就可以窃取完整的密钥

- 可以理解为：一个有一把钥匙的保险柜，并把一个钥匙分成多个部分；需要把钥匙的多个部分合并为一个完整的钥匙后，拿着这个完整的钥匙才能打开保险柜



SSS Vault: One Key Split in Two, One Lock. The vault looks the same as Classic Vault

- Shamir密钥分享算法SSS 简析: <https://blog.mythsman.com/post/5d2fe659976abc05b345449c/>
- Shamir密钥分享算法的核心原理: k个点可以唯一确定一个最高为k-1次的多项式

算法思路

表示

Shamir密钥共享算法由一个二元数 (k, n) 表示，其中 n 表示将明文 s 加密为 n 个Shadow， k 表示必须至少同时拥有 k 个Shadow才能解密获得成明文。

加密

对于待加密的明文 $s \in \mathbb{Z}_p$ (p 为大素数)，在有限群 $GF(p)$ 任取 $k-1$ 个随机数 a_1, a_2, \dots, a_{k-1} ，并令 $a_0 = s$ ，从而构造如下的多项式：

$$f(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + \dots + a_{k-1}x^{k-1} \bmod(p)$$

多项式中只有 p 是公开的，其他 a_i 都是不对外公开的

对于这个多项式，任取 n 个数 $x_1, x_2, x_3, \dots, x_n$ 分别带入多项式得到 n 个密钥对：

$$(x_1, f(x_1)), (x_2, f(x_2)), (x_3, f(x_3)), \dots, (x_n, f(x_n))$$

分发给 n 个持有者。

解密

假设得到了 k 个密钥对 $\{x_1, y_1\}, \{x_2, y_2\}, \dots, \{x_k, y_k\}$ ，我们可以得到如下方程(运算均在 $GF(p)$)：

$$\begin{aligned} a_0 + a_1x_1 + a_2x_1^2 + \dots + a_{k-1}x_1^{k-1} &= y_1 \\ a_0 + a_1x_2 + a_2x_2^2 + \dots + a_{k-1}x_2^{k-1} &= y_2 \\ &\dots\dots\dots \\ a_0 + a_1x_k + a_2x_k^2 + \dots + a_{k-1}x_k^{k-1} &= y_k \end{aligned}$$

由矩阵乘法或者Lagrange插值法均可求的 a_0 即为明文 s 。

安全性

由伽罗华域以及矩阵运算的性质可知该算法的安全性是显然的。

补充

当 $k = n$ 的时候，Shamir算法还有一种用异或运算的实现：任取 $n-1$ 个随机数 $(r_1, r_2, r_3, \dots, r_{n-1})$ ，对于明文 s 计算

$$r_n = r_1 \oplus r_2 \oplus r_3 \dots \oplus r_{n-1}$$

这样就可以通过将这 n 个数全部进行异或来得到明文，同时，任意一个Shadow都不会泄露有关秘密的任何信息。

- [shamir 门限 分散图文详解](#)

3. 阈值签名：从外侧看起来，与普通的私钥一样，外部人员是看不出它不一样的

- **keys are ALWAYS separated** (与SSS是不同的)
- **keys NEVER meet each other** (与SSS是不同的)
- 可以理解为：一个有一把钥匙的保险柜，钥匙的各个部分分布式生成、分布式使用，永远不需要合并到一起；先用一个钥匙碎片旋转一个角度、再用另一个钥匙碎片旋转一个角度才可以打开保险柜。
- 注：MPC和 SSS 是不一样的，SSS虽然分片密钥，但是**最终要重构出密钥来签名**，那么就存在单点故障和重构出的密钥被泄露的可能。而阈值签名不需要重构出密钥。



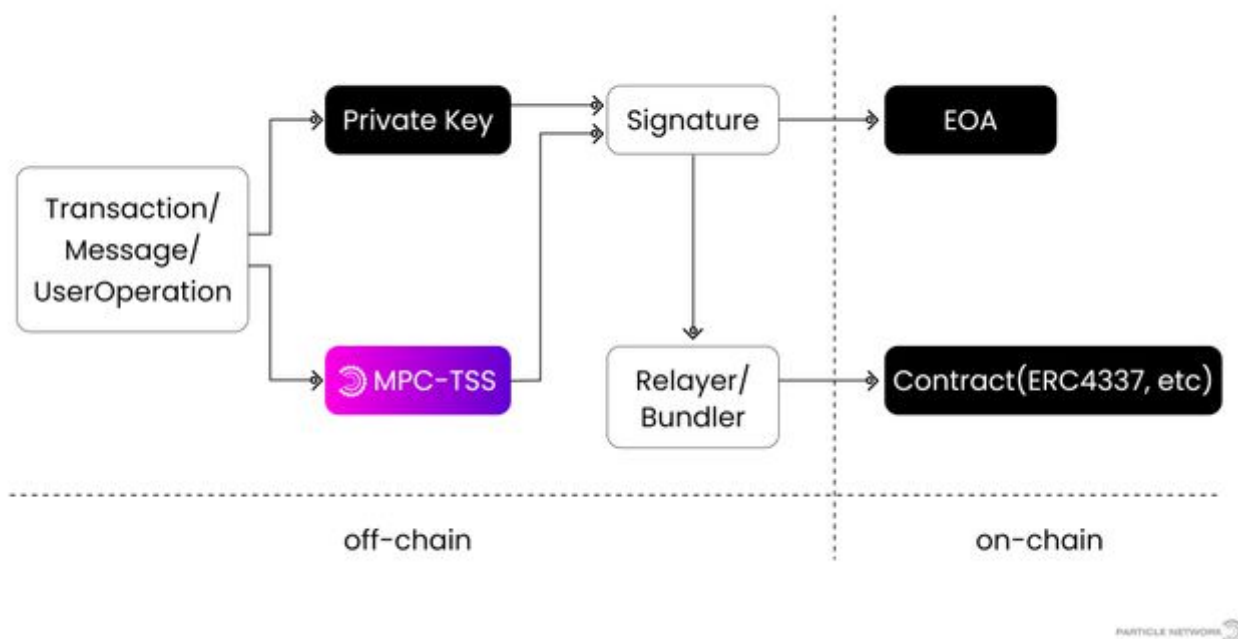
Threshold Signatures Vault: Two Keys, One Lock. The vault looks the same as Classic Vault

MPC钱包 vs 多签钱包

- MPC多方钱包 vs 多签钱包,虽然两者都可以实现“多签”的效果，在签署区块链交易时都涉及多方。但是，存在重要差异。TSS 在链下使用密码学，而多重签名发生在链上。在 TSS 中，签名者的详细信息被折叠到一个看起来很常规的交易中，从而降低了成本并维护了隐私。另一方面，多重签名可以是非交互式的，这省去了在不同签名者之间运行复杂通信层的麻烦。
- 多签钱包：是建立在智能合约上的钱包，同时存在**多方的多对密钥**，需要**多个加密签名来授权交易**。合约中定义了签名的具体验证逻辑，比如验证一笔交易时，需要至少N个私钥中的M个私钥进行签名授权交易等。签名可以来自不同的来源，例如不同的人、不同的机器，甚至同一个人持有的不同加密密钥。
- MPC钱包：将**一个私钥**分解成多个片段，每一方仅持有单个密钥的一部分（或“分片”），而不是唯一的私钥。为了与 MPC 签署交易，多个份额被安全地“计算”成一个有效的密钥，验证过程只涉及到一个私钥。并且整个计算过程是链下执行的，与智能合约并无关系。
- 使用 MPC，重点是对私钥进行分布式计算以签署交易。对于多重签名，重点在于组合多个私钥以签署交易的方法。
- 多签通常是链上进行的，费用较高，并且不同的链多签的实现方案差别很大。而MPC钱包是在链下通过纯密码学的计算生成签名，兼容性更强。
- 多签是可以异步进行签名，而阈值签名要求所有参与方在操作过程中都同时在线。

账户抽象 (Account Abstract) vs MPC

- 两者不是一个维度。如下图所示，MPC处于链下部分、本质是在解决私钥签名相关的问题，对应的签名用来控制链上的 EOA或者智能合约钱包（账户抽象是智能合约钱包的标准化）都可以
- 两者是互补关系。MPC在密钥生成和管理级别提供了共享安全性，而智能合约则为功能和应用程序开发带来了可扩展性和生态系统方法



MPC钱包竞品分析

- 列举市面上的MPC钱包产品，

Safeheron

- <https://www.safeheron.com/zh-CN/mpc-wallet>
- Safeheron 是一个透明、开放的企业级数字资产自托管平台，通过运用 MPC + TEE 等技术提供数字资产领域私钥管理和资产管理解决方案。

ZenGo

- 官网：<https://zengo.com/>
- 开源的多方签名库代码：<https://github.com/ZenGo-X/multi-party-ecdsa>
- 参考资料：<https://zhuanlan.zhihu.com/p/539971122>
- <https://zengo.com/how-zengo-guarantees-access-to-customers-funds/>
- 2018年开始成功运营，从未被黑客入侵；支持MPC、无需助记词；钱包恢复模型；内置web3防火墙
- ZenGo 身份验证需要您控制的 **3 个安全因素**，包括电子邮件、您选择的云和安全的自拍生物识别扫描。如果攻击者可以访问您的电子邮件或您的云恢复工具包，他们将无法恢复和访问您的 ZenGo 帐户，因为还需要生物识别Face ID。ZenGo 提供了一个选项，可以为所有 3 个参数（辅助电子邮件、辅助云恢复工具包和辅助可信自拍）添加辅助选项。



ZenGo 钱包则运用了门限签名方案TSS，它使用两个独立（部分）密钥来取代传统的单个私钥模式。其中一个密钥保存在手机上（用 TouchID/FaceID 授权访问）---Client Share；另一个存储在 ZenGo 服务器上--- Server Share。同时每个客户端还会获得该Server Share的加密版本（使用ZenGo的主密钥加密后的结果，主密钥由ZenGo保管、对应的解密密钥托管在EscrowTech）

Client Recovery（恢复Client Share）：当设备丢失时，ZenGo 钱包提供了一个对设备部分的**密钥备份**的方案：设备密钥通过加密之后存储在 ZenGo 服务器上，而对应的解密密钥则单独存储在您的个人云帐户（例如，iCloud/google drive/dropbox）中，只有通过生物特征认证等授权后才能恢复解密密钥。因此只要设备丢失和icloud关停两件事情不同时发生，就可以还原出设备部分密钥，从而取回资产。

恢复Server Share（CHILL STORAGE™）：ZenGo 构建了第三方独立的托管服务 Escrow（ZenGo服务器的一个备份）和 监听服务Trustee（监听 ZenGo 服务的状态）。当 Trustee 发现ZenGo 服务关停时，它会请求 Escrow 把对应的密钥转移到每个用户Github账号。这种情况ZenGo钱包（客户端）会进入**恢复模式**，从而还原出**完整私钥**，这个私钥可以直接进行交易签名进行资产转移，也可以把这个私钥导入到其他的钱包。具体过程可参考：<https://zengo.com/how-zengo-guarantees-access-to-customers-funds/>

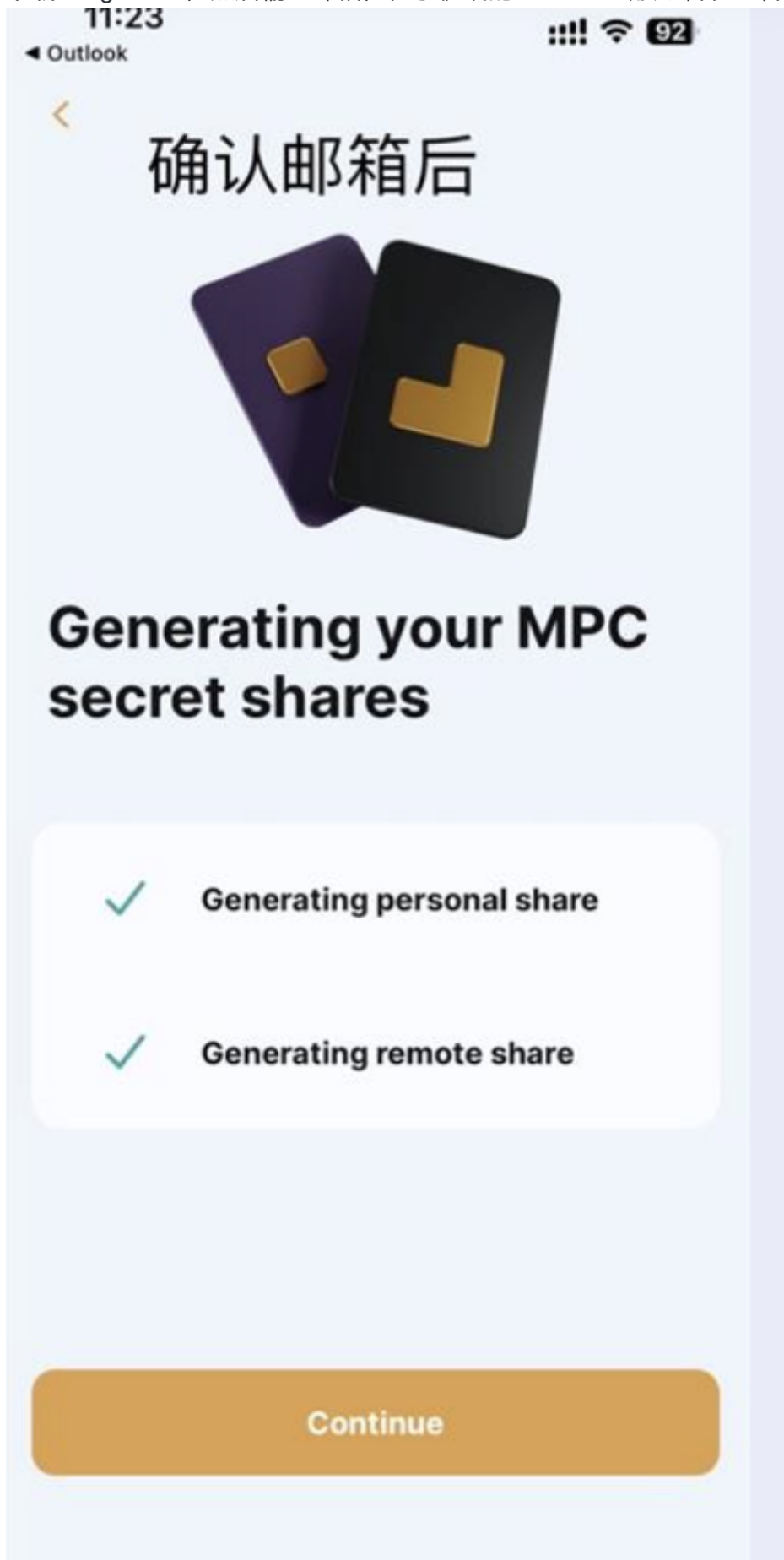
ZenGo 使用分布式安全，我们的服务器通过它持有一个密钥共享（“服务器共享”）来签署交易，而另一个共享在用户的移动设备上（“客户端共享”）。只有当双方的股票相互作用时，交易才能发生。这意味着 ZenGo 无法访问客户的资产。

在正常的 ZenGo 操作期间，每个共享由服务器或客户端保存，永远不会透露给另一方。但是，当向同一方披露两个共享时，可以计算出对应于由共享控制的地址的私钥。使用 CHILL STORAGE™ 解决方案，如果 ZenGo 服务停止运行，对应于客户端共享的服务器共享将显示给客户的应用程序，然后计算私钥。

ZenGo仅用来做交易签名，服务器和设备进行通信签署交易，通信过程不会相互暴露密钥，仅仅是对签名后的数据进行通信。

ZenGo注册流程

1. 下载Zengo APP，然后输入邮箱，在手机端的outlook上确认邮件。邮件确认后就会生成两个密钥分片



2. 然后授权使用 TouchID/FaceID ---用于访问 Client Share



Fireblocks

Lit

Lit是一个去中心化协议，它将密钥分片存储在Lit网络节点上。在这里，公钥/私钥对由PKP（可编程密钥对）NFT表示，其所有者是密钥对的唯一控制者。然后，PKP所有者可以触发网络聚合密钥分片，以解密文件或在满足任意定义的条件时代表密钥分片签名消息。

Qredo

coinbase wallet

my

参考项目（Go版本）：<https://github.com/taurusgroup/multi-party-sig>

为防多点故障，那么就是门限签名。如果是私钥保护，那么就可以同态加密的方式
两步签名+迪菲-赫尔曼密钥交换+零知识证明安全通道

- 安全计算（安全多方计算）：总的来说，大致可归为两类：一类是基于噪音的，另一类不是基于噪音的。
- 1. 基于噪音的安全计算方法，最主要代表是目前很火的差分隐私（differential privacy）。这类方法的思想是，对计算过程用噪音干扰，让原始数据淹没在噪音中，使别有用心者无法从得到的结果反推原始数据。这就好像我们拿到一张打了马赛克的图片，虽然可能可以猜出马赛克后面大概长啥样，但很难知道马赛克后面的所有细节。可参考：<https://zhuanlan.zhihu.com/p/40760105>
- 2. 非噪音方法一般是通过密码学方法将数据编码或加密。这一类方法主要包括三种：**混淆电路**（Garbled Circuit）、**同态加密**（Homomorphic Encryption）和**密钥分享**（Secret Sharing，包含(t,n) Shamir阈值密钥分享）。这些方法一般是在源头上就把数据加密或编码了，计算操作方看到的都是密文
 - 同态加密：将数据加密后，对加密数据进行运算处理，之后对数据进行解密，解密结果等同于数据未进行加密，并进行同样的运算处理。目前，同态加密支持的运算主要为加法运算和乘法运算。按照其支持的运算程度，同态加密分为半同态加密（Partially Homomorphic Encryption, PHE）和全同态加密（Fully Homomorphic Encryption, FHE）
相较于混淆电路和秘密分享，同态加密的思路其实更为直观：直接将原文加密，然后在密文上进行各种运算，最终得到结果的密文，形式化的表示为：

$$x_1, x_2, \dots, x_n \rightarrow [x_1], [x_2], \dots, [x_n]$$

$$f([x_1], [x_2], \dots, [x_n]) \rightarrow [f(x_1, x_2, \dots, x_n)]$$

- 同态加密的应用：云计算领域：一个用户想要处理一个数据，但是他的计算机计算能力较弱。这个用户可以使用云计算的概念，让云来帮助他进行处理而得到结果。但是如果直接将数据交给云，无法保证安全性啊！于是，他可以使用同态加密，然后让云来对加密数据进行直接处理，并将处理结果返回给他。

- 门限签名TSS方案中目前应用最广泛且最成熟的是：GG20算法（其中使用了Shamir秘密共享、零知识证明等）

[零基础向科普之门限签名](#)