门限 ECDSA 密码算法库白皮书

Antalpha

2023.7

目录

1 背景

MPC，即安全多方计算（Multi-Party Computation），是一种加密协议，可以帮助解决数字货币的私钥安全问题。私钥是数字货币所有权和交易验证的关键组成部分，因此保护私钥的安全至关重要。

MPC 使用分散的方式将私钥拆分为多个部分，并将其分发给参与者，每个参与者只掌握其中一部分私钥的信息。这样，任何一个参与者单独的私钥部分都是不完整和无效的，只有当所有参与者共同合作时，才能恢复出完整的私钥。

MPC 协议还使用安全多方计算技术，确保在计算过程中不会泄漏私钥的任何信息。具体来说，MPC 参与者在协议的执行过程中相互通信，每个参与者只能看到他们自己的输入和输出，而不能获得其他参与者的私密信息。这种方式保证了私钥的机密性。

通过使用 MPC，私钥被保护在多个参与者之间，并且需要集体行动才能恢复出完整的私钥。这大大降低了私钥被盗取或滥用的风险，提高了数字货币的安全性。

总结起来，MPC 协议通过将私钥拆分为多个部分，并使用安全多方计算技术在参与者之间进行私密信息的共享和计算，来解决数字货币的私钥安全问题。

本白皮书介绍了我们使用的高安全的 MPC 算法来解决私钥安全问题，并且在算法的软件实现过程中，如何在安全性、功能性、性能方面进行设计，保证最终算法库的安全性、可用性。

2 目标

编写安全密码算法库的目标是提供强大、可靠和经过充分验证的密码算法。我们的目标是确保密码算法库的安全性，包括：

1. 抵御常见攻击：密码算法库必须能够防止常见的攻击，例如穷举、字典、碰撞和侧信道攻击。安全协议必须是安全的，在安全假设上，考虑半诚实模型、恶意模型等攻击场景。

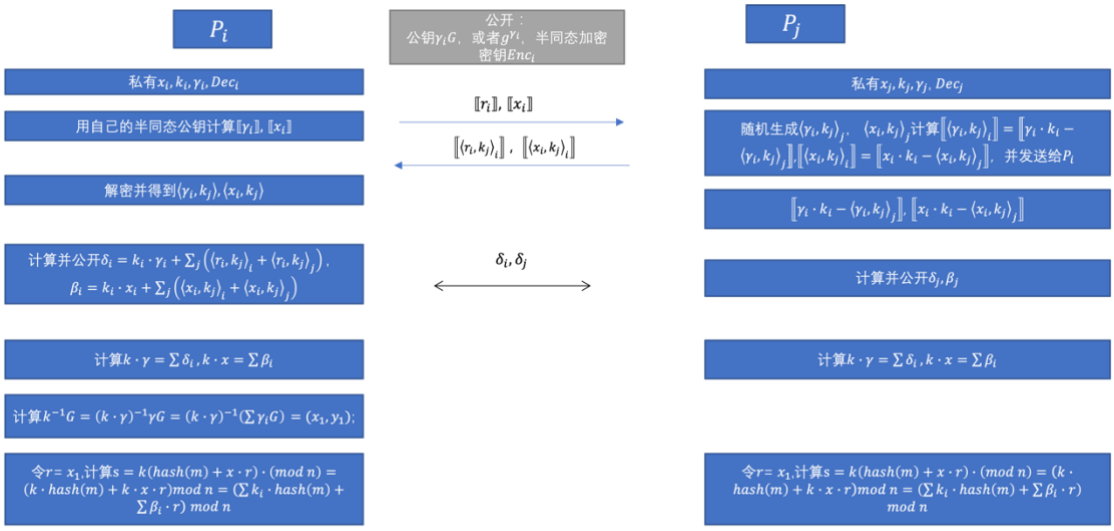
2. 弹性和灵活性：密码算法库应支持不同的密码策略，并在需要时能够轻松扩展或修改。

3. 开放和透明：密码算法库的设计和实现应该是开放和透明的，以便独立的安全专家可以审查和验证其安全性。

4. 最佳实践的遵循：密码算法库应遵循最佳实践，如使用强大的密钥和随机数生成器，进行适当的输入验证和过滤，以及遵循密码学原则。

3 算法安全性设计

3.1概述



1 MPC 密钥分片采用 Shamir 密钥共享方案，使用$(\boldsymbol{t}, \boldsymbol{n})$门限,并且采用 Feldman's VSS 进行 ZK 承诺与验证

2 MPC 分片的加法分享采用拉格朗日插值进行加法分片与组合

3 签名过程中的 k 跟前面的消息没有相关性，可以对 k 相关的计算进行预计算

4 在进行乘法运算的秘密分享时，采用乘法转加法的方式进行运算

5 每一方使用任何私有的信息进行计算时，都要采用 zk 进行承诺，对方进行 zk 验证，防止参与方的恶意运算或者传输过程中被非法篡改。

具体算法可以参考 *UC Non-Interactive, Proactive, Threshold ECDSA with Identifiable Aborts*。

3.2Shamir 门限方案

一共n个节$(P_1, P_2, \cdots, P_n)$拥有自己的秘密值，如何计算这n个节点秘密的和，即求$v = v_1 + v_2 + v_3 + \cdots + v_n$，如果是门限方案，那么 t 个参与方就可以恢复出$v = v_1 + v_2 + v_3 + \cdots + v_n$

- 步骤一:随机值的共享，每个节点拥有均需一份相同的 $X = \{x_1, x_2, \ldots, x_n\}$,一般情况下取$\{1,2,\cdots,n\}$
- 步骤二:各节点P_j分别随机选择一个$t-1$次的多项式$f_i(x), (f_i(x) = a_i x^{t-1} + b_i x^{t-2} + \cdots . + v_i)$，因此这里加上 v_i 在内一共产生了t个未知数$(a_i, b_i, \cdots, v_i)$，这t个数是只有当前P_j才会知道的，且每个节点都会随机产生不同的t个未知数，其中$t-1$个随机数，1 个秘密值。
- 步骤三:各节点 P_j根据自己的多项式f和随机值 X确定一组共享$f_1(x_i)$，$f_2(x_i)$，$f_3(X_i), \cdots$，$f_i(x_n)$,分别发给节点 $P_1, P_2, \cdots, P_n$。

- 步骤四:各节点P_j将自己收到的结果($f_1(x_i)$、$f_2(x_i)$、$f_3(x_i)$、…、$f_n(x;)$)相加得到$R_i=(a_1+a_2+a_3+\cdots+a_n)x^{t-1}+(b_1+b_2+\cdots+b_n)x^{t-2}+\cdots+(v_1+v_2+v_3+\cdots+v_n)$，并将结果发送给其他节点。

- 步骤五:节点收到的$R_i(i=0,1,\cdots,t)$放在一起就是一个t元[$(a_1+a_2+a_3+\cdots.+a_n)$、$(b_2+b_3+\cdots+b_n)$、…、$(v_1+v_2+v_3+\cdots+v_n)$]的方程组，节点收到多于等于$t$个节点的$R_i$就能解出$(v_1+v_2+v_3+\cdots+v_n)$。

3.3 Feldman's VSS

为了检验共享的正确性,出现了可验证的秘密共享方案。 一个正常执行的可验证秘密共享方案能够保证:在秘密分发阶段，分发者发送给参与者的共享是正确的;在秘密恢复阶段，参与者提交的共享也是正确的

$$f_i(x)=a_ix^{t-1}+b_ix^{t-2}+\cdots.+v_i$$

P_i对a_i，　b_i都做了承诺$A_i=g^{a_i},B_i=g^{b_i}$，$V_i=g^{v_i}$，$x_i\in[1,t]$,可用下式进行验证

$$g^{f_i(j)}?=g^{a_ij^{t-1}+b_ij^{t-2}+\cdots.+v_i}=A_i^{j^{t-1}}B_i^{j^{t-2}}\cdots V_i$$

4 算法实现安全性设计原则

为了实现上述目标，我们提出以下设计原则来编写安全的密码算法库:

1. 选择合适的密码算法与密码策略：密码算法库应该使用经过验证和广泛使用的密码算法，如 AES、RSA 和 SHA 等。根据需求和安全级别选择合适的密码算法和密钥长度。密码策略方面提供灵活的密码策略，包括密码复杂性要求、密码过期和重置等。
2. 基础密码库的选择：选择经过业界长期验证的算法库，如 open SSL、gmp、crypto++、seal 等。

3. 密钥管理和保护：密码算法库应提供安全的密钥管理和保护机制，包括密钥生成、存储、传输和销毁等方面。

4. 随机数生成器：密码算法库应使用强大和可信的随机数生成器，以提供有足够的熵来生成安全的密钥和随机值。需要通过 NIST SP 800-90A/B/C 、AIS 20/31 真随机数测试标准以及 NIST SP 800-90A 的确定性随机数标准。

5. 输入验证和过滤：密码算法库应该对输入进行适当的验证和过滤，以防止缓冲区溢出、注入攻击和其他安全问题。

6. 安全性测试：密码算法库应经过充分的安全性测试，包括代码审查、单元测试、集成测试和漏洞扫描等。

7 错误处理：在密码算法库中实现适当的错误处理和异常处理机制，避免信息泄露和安全漏洞。

8. 文档和示例：提供清晰、准确和易于理解的文档和示例，以帮助开发人员正确使用密码算法库。

9. 生命周期管理：持续更新和维护，不断跟踪最新的安全标准和技术发展，并及时更新和维护密码算法库。

5 时间攻击与侧信道攻击

时间攻击和侧信道攻击都是针对计算机系统的安全性的一种攻击手段。

1. 时间攻击（Timing Attack）：

时间攻击是一种基于计算机系统操作时间或时间差异的攻击方法。它利用了算法或程序在不同输入条件下执行时间的差异，通过观察这些差异来获取敏感数据或破解密码。例如，一个网站可能会有一个登录功能，如果用户提供的用户名和密码与存储的凭据匹配，则会给出一个成功登录的响应。但是，由于算法在比较输入值时使用了不同的时间，攻击者可以通过多次尝试不同的输入组合，根据服务器发回响应所需的时间来获得关于正确凭证的信息。

如果为了防范时间攻击，开发人员需要确保程序在处理敏感信息时保持固定的运行时间。这可以通过使用恒定时间比较函数、随机化延迟等技术手段来实现。

2. 侧信道攻击（Side-Channel Attack）：

侧信道攻击是一种利用计算机系统实际执行过程中泄露的额外信息来推断出敏感数据的攻击方法。这些额外信息可以是电压、电磁波辐射、功耗、甚至是声音等。通过监测这些信道，攻击者可以推断出密钥、密码或其他敏感信息。例如，功耗侧信道攻击利用密码设备运行时的功耗变化来提取密钥信息，电磁辐射侧信道攻击利用电磁辐射来提取密钥信息，声音侧信道攻击利用声音来提取密钥信息。

国际 CC 认证标准中定义了 7 种安全等级，其中 CC4+定义的 VAN.5 安全等级，定义了一个设备对侧信道攻击的防御程度的技术上的最高标准。

算法在设备的实际运行中，ECC 标量乘跟 RSA 模幂在运行时，通常采用窗口法等算法，其在运算过程跟密钥不一样，那么运算时间跟功耗都是不一样的，如果是在智能卡等存在物理攻击的场景的设备上运行时，需要进行专门的防护设计，但算法如果运行在服务器、PC、手机等设备上，那么只会存在远程的时间攻击威胁，但是因为运算速度很快，只靠运行时间不一样，是无法获得实际的密钥等敏感信息的。

如果需要防护那么在通讯之前，加入一些随机的时间延迟，就是比较好的防护方案。

6 主要算法功能

6.1基础大数运算库

包括包括但不限于 4096bit 的大整数的基础运算，如模加、模减、模乘、模逆、模幂、素数生成、Miller–Rabin primality test 等等。

底层运算支持 gmp 以及 go 语言本身的大数运算库。

6.2ECC 运算

ECC256 密钥生成、点加、标量乘等

6.3ZK

Schnorr、Dlog Equality、Paillier Decryption modulo q、Paillier Multiplication、Paillier 加密下的 Pedersen 范围证明、Paillier Encryption in Range、Dlog with El-Gamal Commitment、Paillier Encryption in Range with El-Gamal Commitment、Dlog vs Paillier Encryption in Range、Multiplication Paillier vs Group Commitment、Paillier Affine Operation with Group Commitment in Range、Paillier Affine Operation with Paillier Commitment in Range、OR proof、No Small Factor Modulus、N-th root

6.4ElGamal 算法

密钥生成、加密、解密

6.5Paillier 算法

密钥生成、加密、解密、同态加法、标量乘法

6.6普通 ECDSA

ECC 密钥生成、签名、验证

6.7门限 ECDSA

门限密钥生成、门限密钥刷新、门限签名的预签名、直接门限签名、签名验证、
门限密钥包括随机数密钥以及助记词密钥

6.8TLS 整数认证以及加密通讯

TLS 身份认证采用二级证书认证，通信采用了 TLS 生成的会话密钥进行

7 算法库使用安全实践建议

为了确保密码算法库的安全性，利用算法库进行应用开发，如进行钱包开发时，我们我们提供以下安全设计建议：

1. 输入验证和过滤：确保用户输入的数据符合预期的格式和范围，并且不包含恶意代码或特殊字符。在 Go 中，可以使用正则表达式或库函数来进行输入验证和过滤。

2. 防止注入攻击：避免直接将用户输入的数据拼接到 SQL 查询、命令行参数或系统调用中，以防止 SQL 注入、命令注入或代码注入等攻击。应该使用预编译语句、参数化查询或使用 ORM 库来处理数据库访问。

3. 跨站脚本攻击（XSS）：防止恶意用户将 HTML、JavaScript 或其他客户端脚本注入到应用程序的输出中。在 Go 中，可以使用模板引擎、自动转义或过滤用户输入来防止 XSS 攻击。

4. 跨站请求伪造（CSRF）攻击：确保每个请求都是经过授权的，通过实施合适的 csrf token 验证并将其与用户会话绑定。在 Go 中，可以使用反向代理、中间件或专门的 CSRF 保护库来防止 CSRF 攻击。

5. 密码安全：存储密码时，应使用散列函数和盐值对密码进行加密，以防止密码泄露后被破解。推荐使用 bcrypt 或 scrypt 等密码哈希算法。

6. 敏感数据保护：应该对敏感数据（如密码、API 密钥和数据库凭据）进行加密存储，并在传输过程中使用加密协议，如 HTTPS。

7. 访问控制和权限管理：确保只有授权用户可以访问特定的资源和功能。在 Go 中，可以使用身份验证和授权库来处理访问控制和权限管理。

8. 日志和监控：记录应用程序的日志并实时监控应用程序的行为，以便能够及时检测和响应潜在的安全问题。

9. 依赖管理和更新：及时更新应用程序使用的第三方库和框架，以修复已知的安全漏洞。使用可靠的依赖管理工具，如 Go Modules。