初识TON：账号、Token、交易与资产安全

By: Johan

### 背景

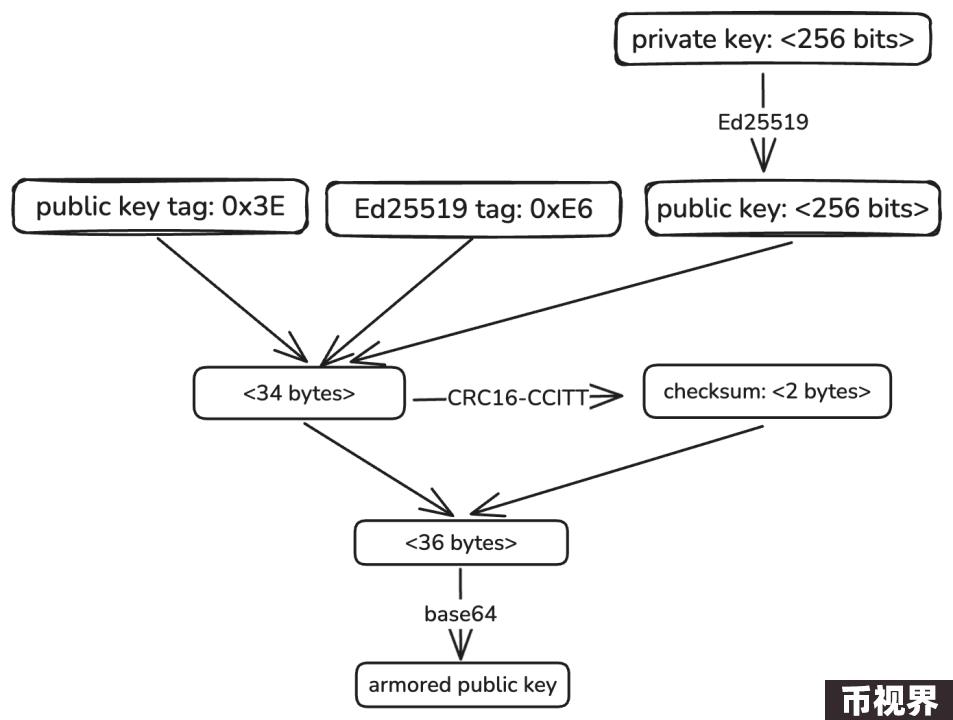
TON(The Open Network) 是一个去中心化的区块链平台，由 Telegram 团队最初设计和开发。TON 的目标是提供一个高性能和可扩展的区块链平台，以支持大规模的去中心化应用(DApps) 和智能合约。

TON 如此特殊，它是易用的，它与 Telegram 深度结合，使得普通人也能轻易使用代币；它又是复杂的，它与其他区块链有着截然不同的架构，而且使用非主流的 FunC 智能合约语言。今天我们从账号、Token、交易的角度讨论一下 TON 的特点及用户资产安全问题。

### TON 的特点

#### 账号生成

TON 账号地址的生成方式与大多数区块链都不同，它是一个智能合约地址。首先，开局一个私钥，TON 主要使用 Ed25519 算法生成公钥，生成流程如下：

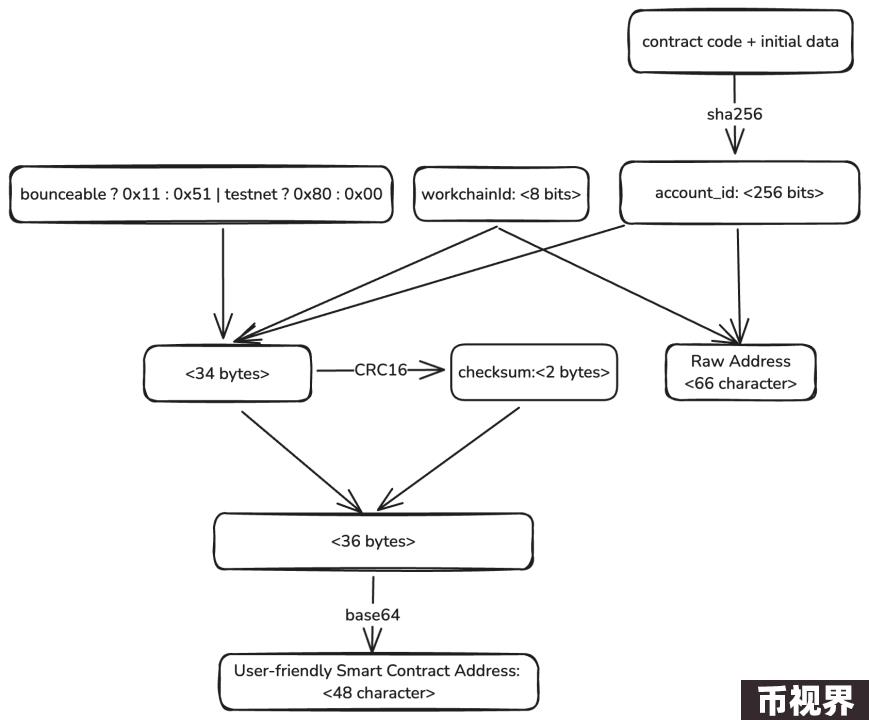


这里有两种形式的公钥，一种是由私钥计算出的原始公钥，形如：

E39ECDA0A7B0C60A7107EC43967829DBE8BC356A49B9DFC6186B3EAC74B5477D

另一种是“美化”后的公钥，它携带了公钥的一些信息和校验位，形如：Pubjns2gp7DGCnEH7EOWeCnb6Lw1akm538YYaz6sdLVHfRB2

如果认为得到公钥就能像以太坊那样得到账号地址就太天真了，仅仅有用户的公钥还不足以计算出用户的账号地址。我们刚刚说了，用户的账号地址是一个智能合约地址，但是我们连账号都没有，怎么部署智能合约？正确的顺序是先计算地址，接收一点初始金额的代币，然后就可以部署合约。账号地址的计算过程如下图所示：



用户的地址也有多种形式，首先是原始形式，形如：

0:b4c1b2ede12aa76f4a44353944258bcc8f99e9c7c474711a152c78b43218e296

以及用户友好形式，形如：

Mainnet: Bounceable: EQC0wbLt4Sqnb0pENTlEJYvMj5npx8R0cRoVLHi0MhjilkPX Non-bounceable: UQC0wbLt4Sqnb0pENTlEJYvMj5npx8R0cRoVLHi0Mhjilh4STestnet: Bounceable: kQC0wbLt4Sqnb0pENTlEJYvMj5npx8R0cRoVLHi0Mhjilvhd Non-bounceable: 0QC0wbLt4Sqnb0pENTlEJYvMj5npx8R0cRoVLHi0MhjilqWY

细心观察这几个地址就可以看出，它们只有首尾几个字符存在差别，中间的 `account\_id` 是一样的，但是我们还是无法看出公钥和账号地址之间的关系，其实玄机就藏在开头的 `initial data` 中，它包含了一个用户的公钥，用户就是通过它掌握钱包合约的所有权。`workchainId` 很好理解，TON 并不是只有一条单链，它由非常多的分片组成，每个分片是整个网络的一部分，处理特定的一组账号和交易。为了定位和管理智能合约，需要明确指出它们位于哪个分片中。`Bounceable` 和 `Non-bounceable` 有什么区别？这和智能合约的工作机制相关，我们先继续往下看。

#### 钱包合约

以下是一个用户钱包合约的一段源代码，可以看到它在接收到用户的消息时读取了 4 个参数(stored\_seqno, stored\_subwallet, public\_key, plugins)：

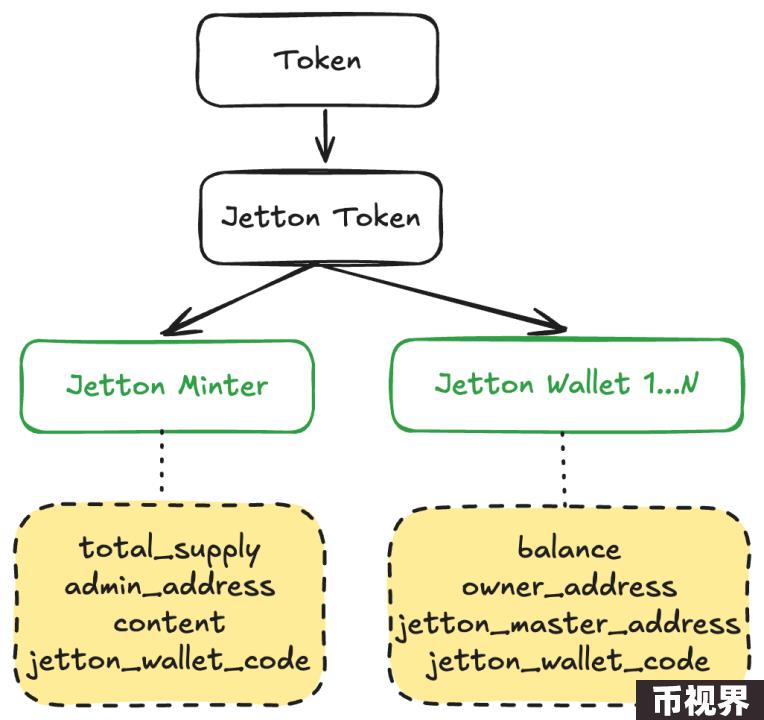
wallet-v4-code.fc

() recv\_external(slice in\_msg) impure { var signature = in\_msg~load\_bits(512); var cs = in\_msg; var (subwallet\_id, valid\_until, msg\_seqno) = (cs~load\_uint(32), cs~load\_uint(32), cs~load\_uint(32)); throw\_if(36, valid\_until <= now()); var ds = get\_data().begin\_parse(); var (stored\_seqno, stored\_subwallet, public\_key, plugins) = (ds~load\_uint(32), ds~load\_uint(32), ds~load\_uint(256), ds~load\_dict()); ;;##The Initial Data ds.end\_parse(); throw\_unless(33, msg\_seqno == stored\_seqno); throw\_unless(34, subwallet\_id == stored\_subwallet); throw\_unless(35, check\_signature(slice\_hash(in\_msg), signature, public\_key)); //...}

没错，这个用户的钱包合约在部署的时候，需要传入一些初始参数，其中就包含了一个 256 位的 public\_key 信息，这样就确保了每个用户使用相同的合约代码时，有一个独立的合约地址。用户发起的一切交易都需要对 `in\_msg` 签名，然后通过自己的钱包合约验证签名(check\_signature) 后，由合约去调用链上的一切操作。从这里我们也可以推断出，一个用户的公钥其实是可以对应无数钱包地址的，只需要部署不同源代码的钱包或者不同的初始化数据，就能得到完全不同的合约地址。

#### Jetton Token

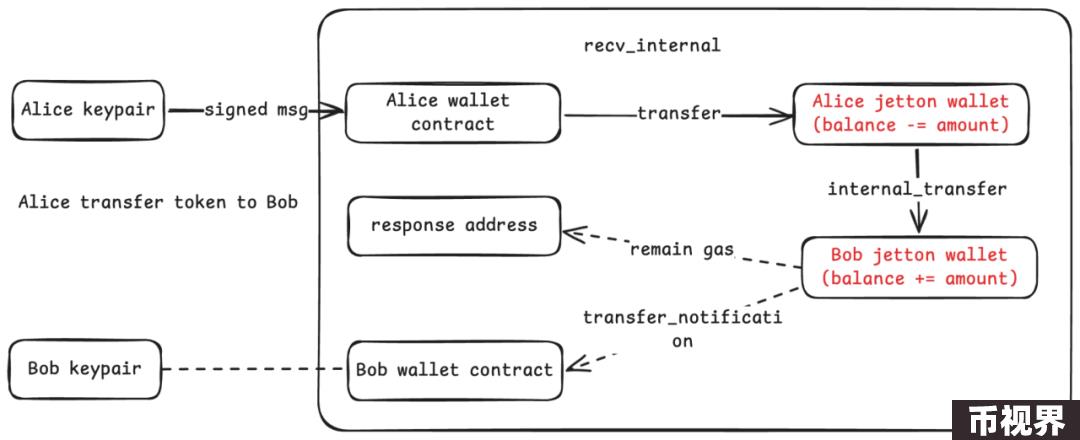
Token 是资产在链上的表现形式，所以它是我们需要了解的一个基本元素。Jetton 是 TON token 的标准形式，Jetton 由两部分合约组成，Jetton-minter 和 Jetton-wallet：



代币被发行时，会创建一个 Jetton-minter 合约，合约初始化记录了代币的总量、管理员、钱包代码等信息。

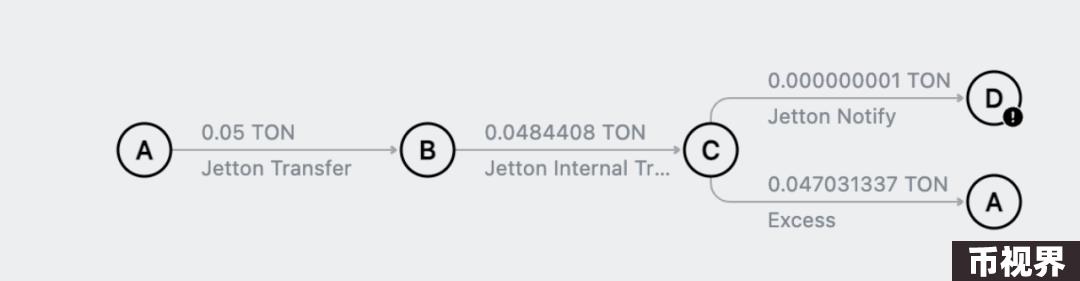
代币被分发给用户时，Minter 合约会为用户部署一个钱包合约，并在合约初始化时记录用户的余额、所有权、代币 Minter 合约地址、用户钱包代码等信息，每个用户都会独立部署一个合约。注意，这里创建的合约是用于管理特定 Jetton 代币的钱包合约，与用户的账号钱包合约并不相同，这里的 owner\_address 记录的是用户的账号钱包地址。

当用户 Alice 给用户 Bob 转账时，调用关系如下：



Alice 在链下的 APP 签名，并通过调用她的钱包合约下达操作指令。这些指令会进一步调用她的代币钱包进行转账。当 Bob 的代币钱包收到代币后，它会通知 Bob 的钱包合约（即 Bob Jetton-wallet 的 Owner 地址）。如果交易过程中有剩余的 Gas，还会返回给响应地址，通常为 Alice 的账号合约。

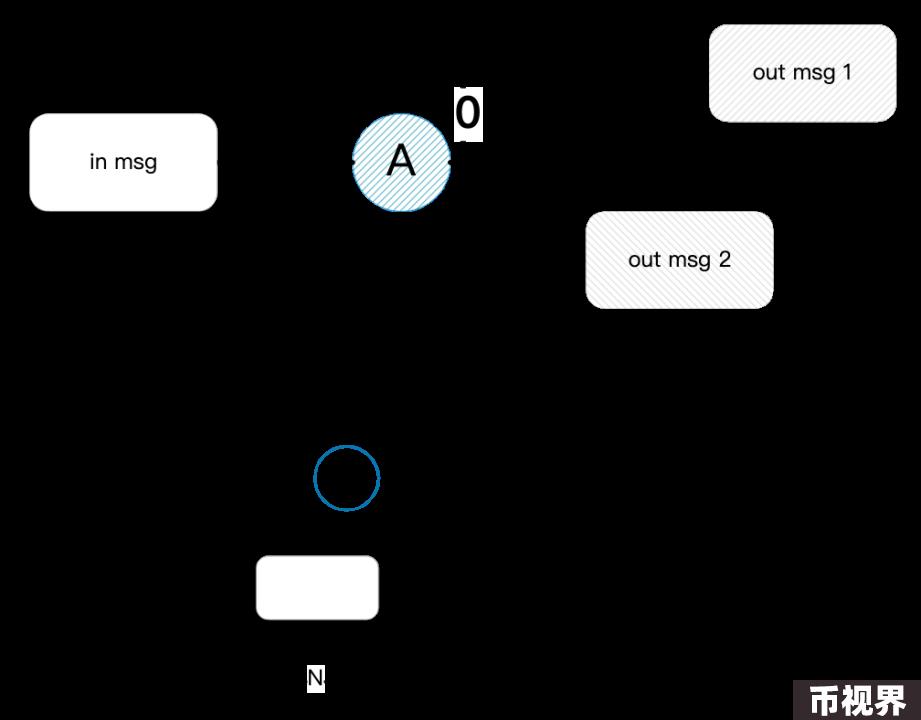
这是 Tonviewer 浏览器解析的一笔 Jetton 代币转账：



一个 ERC20 转账最少只需要调用一个合约，而一笔 Jetton 代币转账至少要调用四个合约，这么做是为了让转账可以在链上并发进行，提高交易效率。

#### 交易

当 TON 中的某个帐户发生某些事件时，它会引发一笔交易，最常见的事件是“接收某个消息”，交易包括以下内容：



交易需要注意几个特性：

1. 异步：TON 的交易并不是在一次调用里完成的，它可能需要通过消息传递到多个不同的智能合约去执行一系列调用。由于分片链中的路由不同，TON 并不能保证多个智能合约之间的消息传递顺序。

2. 手续费:异步的特性还会带来一个问题，即消耗的手续费难以预估。因此在发起交易时，钱包通常会多发送一些代币做为手续费。如果调用的合约有良好的手续费处理机制，那么剩余的手续费最后会返还到用户钱包。用户可能会观察到钱包代币突然变少，过几分钟又变多，就是这个原因。

3. 反弹：反弹是合约的一种错误处理机制，当调用的合约不存在或者抛出错误时，如果交易设置为可反弹的，那么就会反弹回一个 bounced 消息给发起调用的合约。例如：用户发起一笔转账，如果调用过程出错了，那么就需要反弹消息，这样用户的钱包合约才能将自己的余额恢复。几乎所有在智能合约之间发送的内部消息都应该是可弹回的，即应该设置它们的“bounce”位。

### 资产安全

TON 有很多特性会带来安全问题，因此用户也需要了解一些常见的陷阱。

#### 手续费截留攻击

上面说到钱包经常需要多发送一些手续费以防止交易执行失败，这让攻击者找到了作恶的机会。如果你是一个 TON 钱包用户，你可能碰到过这样的情况，钱包里总是会收到各种 NFT 或者代币，本以为只是一些垃圾代币空投，但是一查交易信息，竟然可以卖不少钱？可是当发起交易时，发现需要的手续费超高（1 TON），这时就需要注意了，这可能是手续费诈骗。



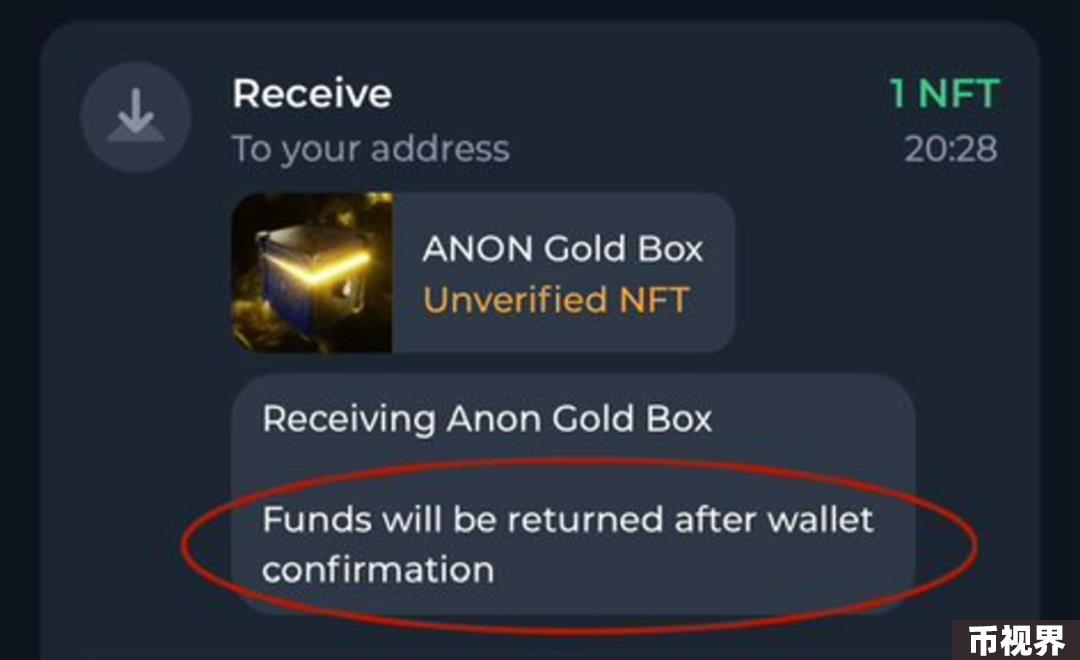
攻击者利用精心构造的代币合约，让钱包的预估转账手续费超高，而实际执行时却只是截留手续费，并未发送转账消息。

#### 首尾号钓鱼

首尾号钓鱼不是 TON 上才有，各大公链都存在这种钓鱼攻击。攻击者会为全网每个用户地址都生成一个首尾号相同的高仿账号，当用户发送一笔转账时，攻击者用高仿账号也尾随发送一笔小额转账，目的是在用户的收款记录里留下一个记录。当收款用户想要转回一笔代币时，可能会从历史记录里复制地址，这时很可能复制到攻击者的地址，导致转账到错误地址，攻击者可谓是精准拿捏用户的行为了。

#### comment 钓鱼

TON 在转账时可以添加一个 comment，用于备注交易信息，这个功能在交易所充值时会频繁用到，交易所通常会要求用户在充值时备注一下用户 ID。但这个功能经常会被恶意利用，攻击者通过在备注里写入欺诈信息来骗取用户的资产。如图所示：



用户尤其需要注意 Anonymous Telegram Number 这个 NFT，如果用户用 Anonymous Telegram Number 开通了 TG 号，但没开 Two-Step Verification，一旦这个 NFT 被钓走，黑客就可以直接登录目标 TG 号，实施后续的资产盗取及欺骗行为。

#### 智能合约漏洞

智能合约的安全漏洞会导致用户放在智能合约的资金受损，用户在选择项目时需要选择经过良好审计的项目。TON 的智能合约主要使用 FunC 语言来编程，也有使用更高级的 Tact，或者更底层的 Fift，都是原创程度很高的语言。新的编程语言会带来新的安全风险，特别是对开发者而言，要有安全编程的良好习惯，掌握最佳安全实践，并且在部署生产环境之前经过严格的安全审计，限于篇幅，本文暂不讨论合约安全。慢雾安全团队已推出 TON 智能合约安全审计服务，欢迎有审计需求的朋友一起探讨。

#### 假充值攻击

钱包或交易所用户需要注意假充值攻击，通常有两种类型的假充值攻击：

### 总结

本文从 TON 的公私钥创建、钱包合约、Token 的形式、交易特性等角度介绍了 TON 的一些基础的技术原理，同时也探讨了使用 TON 的过程中可能存在的安全问题，希望能给大家的学习带来启发。

参考链接：

https://docs.ton.org/

https://github.com/ton-blockchain/wallet-contract