无私钥以太币

Author : Tri0nes

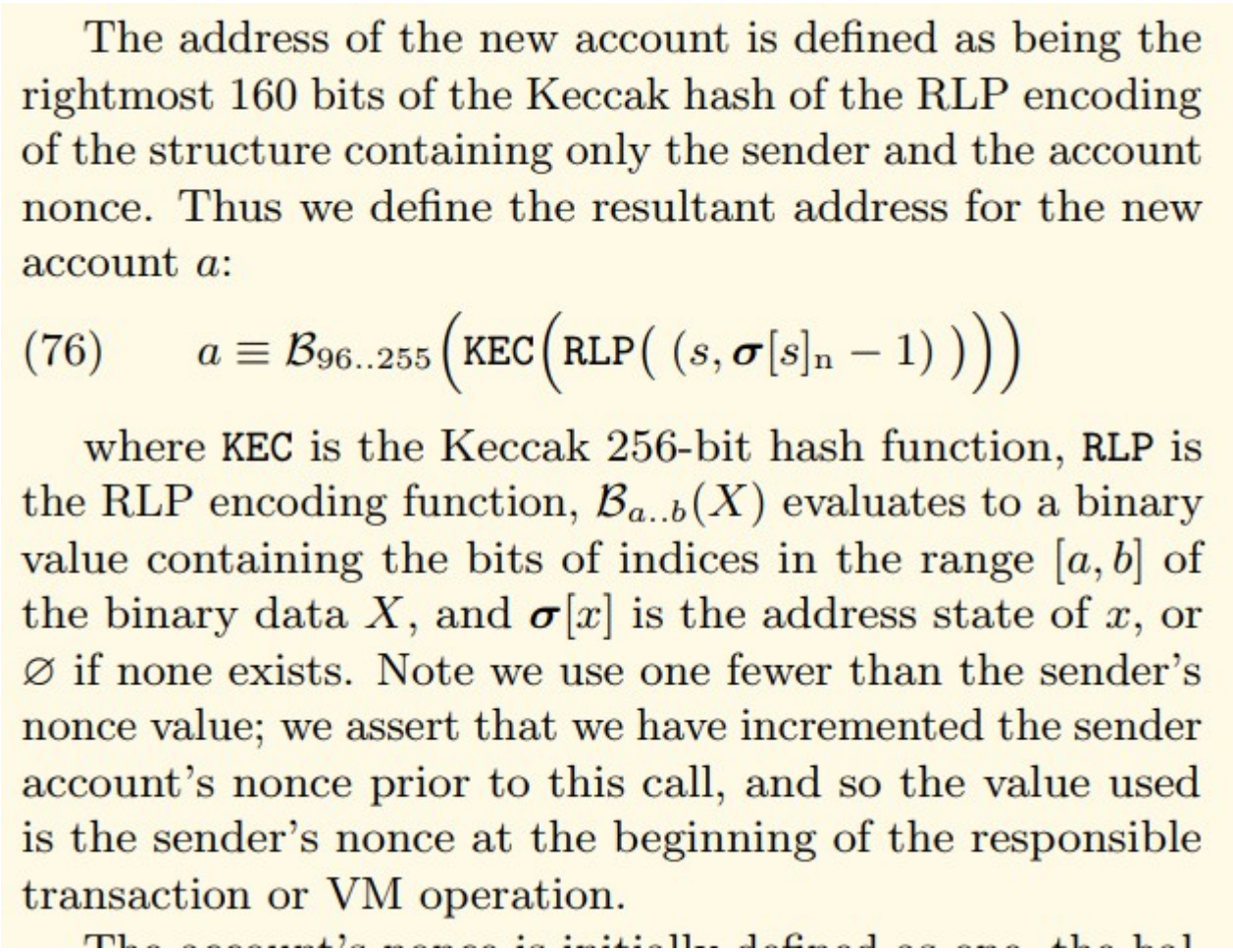
我们都知道以太币存放在账户上，账户都是有私钥的。而今天介绍的无私钥以太币(keyless Ether)则是将以太币放置到一个隐蔽的地址上，即使私钥丢失，你可以找回你的以太币，保证相对"安全"。

一、原理分析

以太坊内有两种类型的账户; 普通账户和合约账户。 合约是通过发送带有空字段的事务(空的 `to`)来创建的，并且包含一些被执行的数据（一个构造函数），并且希望返回一些放在区块链上的代码。 这些合同自然是与正常账户相同的地址空间的一部分; 由此确定合同的地址：

```
address = sha3(rlp_encode(creator_account, creator_account_nonce))[12:]
```

可见黄皮书第7章"创建合约"部分：



The address of the new account is defined as being the rightmost 160 bits of the Keccak hash of the RLP encoding of the structure containing only the sender and the account nonce. Thus we define the resultant address for the new account a:

$$(76) \qquad a \equiv \mathcal{B}_{96..255}\left(\mathbf{KEC}\left(\mathbf{RLP}\left(\ (s, \boldsymbol{\sigma}[s]_{\mathrm{n}} - 1)\ \right)\right)\right)$$

where KEC is the Keccak 256-bit hash function, RLP is the RLP encoding function, $\mathcal{B}_{a..b}(X)$ evaluates to a binary value containing the bits of indices in the range $[a, b]$ of the binary data X, and $\boldsymbol{\sigma}[x]$ is the address state of x, or $\varnothing$ if none exists. Note we use one fewer than the sender's nonce value; we assert that we have incremented the sender account's nonce prior to this call, and so the value used is the sender's nonce at the beginning of the responsible transaction or VM operation.

合约地址是确定性的，由 `keccack256(address,nonce)` 计算。(其中 `address` 是合约的地址(或创建交易的以太坊地址)，而 `nonce` 是合约生产其它合约的一个数值(或者对于常规交易来说是交易的 `nonce`))。

从本质上讲，合约的地址就是账户与交易 nonce 串联的 `keccak256` 哈希值。合约的 `nonce` 是以 1 开始的，账户的交易 `nonce` 是以 0 开始的。

如果你有一个生产合约 `accountA` 地址 `0x6ac7ea33f8831ea9dcc53393aaa88b25a785dbf0`，那么此合约创建的合约的地址将如以下顺序：

```
nonce0= "0xcd234a471b72ba2f1ccf0a70fcaba648a5eecd8d"
nonce1= "0x343c43a37d37dff08ae8c4a11544c718abb4fcf8"
nonce2= "0xf778b86fa74e846c4f0a1fbd1335fe81c00a0c91"
nonce3= "0xfffd933a0bc612844eaf0c6fe3e5b8e9b6c1d19c"
```

从 accountA 我们可以发送 ether 到 nonce1 地址。我们需先创建一个交易(以此增加 accountA 的 nonce 到 1)，然后创建基于这个 `nonce1` 地址的合约，由此合约就能成为这些资金的所有者了，最终把资金转回。

这意味着你可以发送 ether 到预先已经确定的地址（这个地址你不拥有私钥，但你可以在这个地址上创建合约）。在你可以发送 ether 到这个地址之后通过创建能衍生相同地址的合约来回收 ether。合约中的构造函数可以用来返回所有你预先发送的 ether。

这也可以在没有合约的情况下完成。您可以将 `ether` 发送到可以从您的一个标准以太坊帐户创建的地址，并在以后以正确的 nonce 恢复。那么即使有人获取你所有的以太坊私钥，但攻击者很难发现你以这种方法隐藏的 ether 。但如果攻击者发起了太多的交易，你的 nonce 被使用了，那就没有可能去恢复隐藏的 ether 了。

二、生产合约隐藏 ether

示例代码：

```
contract KeylessHiddenEthCreator {
    uint public currentContractNonce = 1; // keep track of this contracts nonce publicly
(it's also found in the contracts state)

    // determine future addresses which can hide ether.
    function futureAddresses(uint8 nonce) public view returns (address) {
        if(nonce == 0) {
            return address(keccak256(0xd6, 0x94, this, 0x80));
        }
        return address(keccak256(0xd6, 0x94, this, nonce));
    // need to implement rlp encoding properly for a full range of nonces
    }

    // increment the contract nonce or retrieve ether from a hidden/key-less account
    // provided the nonce is correct
    function retrieveHiddenEther(address beneficiary) public returns (address) {
    currentContractNonce +=1;
       return new RecoverContract(beneficiary);
    }

    function () payable {} // Allow ether transfers (helps for playing in remix)
}

contract RecoverContract {
    constructor(address beneficiary) {
```

```
        selfdestruct(beneficiary); // don't deploy code. Return the ether stored here to
the beneficiary.
    }
}
```

这个合约允许你存储无密钥的以太(在你不会错误地忽略 nonce 的情况下，这是相对安全的)。 `futureAddresses()` 函数可以用于计算此合约可以产生的前 127 个合约的地址(需要指定 nonce)。如果你发送 `ether` 到其中一个地址，那么就可以通过调用足够次数的 `retrieveHiddenEther()` 来恢复这些 `ether` 。

例如，如果您选择 `nonce=2` （并将 `ether` 发送到关联的地址），则需要调用 `retrieveHiddenEther()` 两次，`ether` 就可以恢复到 `beneficiary` 地址。

测试过程如下：

1 使用 remix 部署上面的合约代码

工厂合约创建为: `0x0567a4e52f9871371788c86e4eff9e04cdeb3bb6`

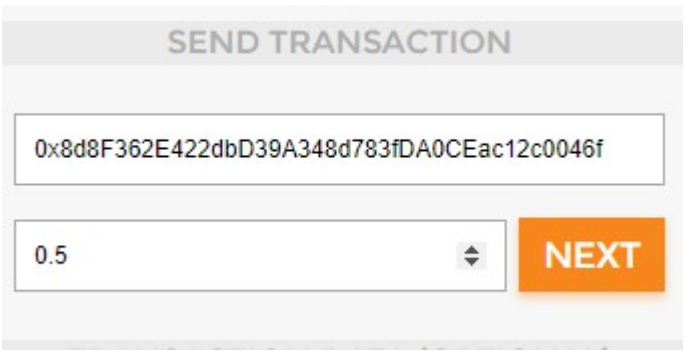
部署工厂合约后查看各个 nonce 值对应的地址。

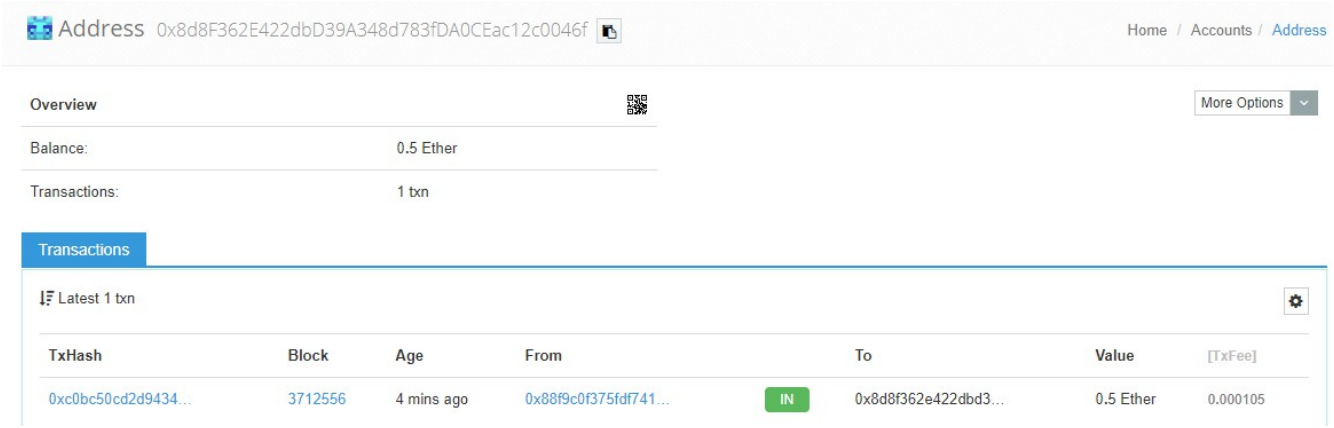
```
nonce0= "0xd3bf9663DdA16B942c406b0A6d579D7CC4A67543"
nonce1= "0x7AB07e42fe62B71029BC40656aA182548B936753"
nonce2= "0x8d8F362E422dbD39A348d783fDA0CEac12c0046f"
nonce3= "0x901e56821F82321B085686f1143039BB0A5B1126"
```

2 隐藏 ether

使用 nonce = 2 计算出我们要隐藏 ether 的地址为 `0x8d8F362E422dbD39A348d783fDA0CEac12c0046f`

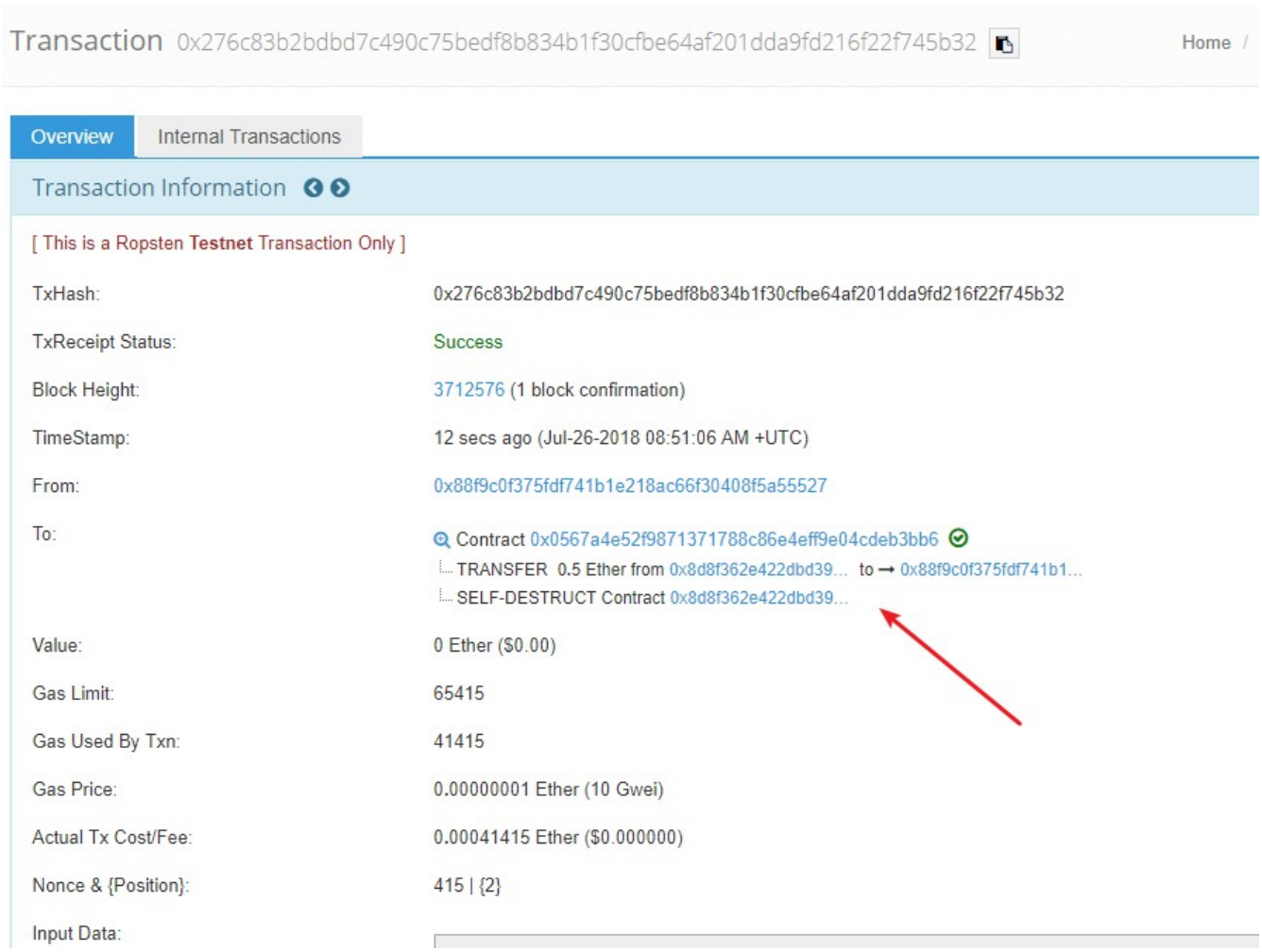
转账:我们先执行了一次转账操作，将 0.5 ether 转到了 nonce2 地址 `0x8d8F362E422dbD39A348d783fDA0CEac12c0046f` 上。



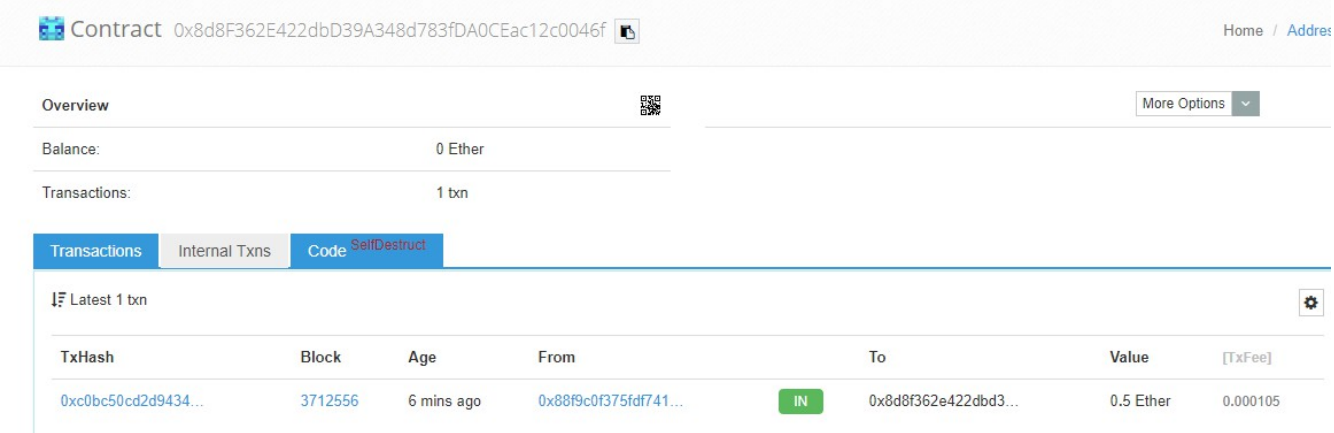


3 恢复 ether

执行 `retrieveHiddenEther()` 两次，其中 `ether` 就可以恢复到 `beneficiary` 地址，如以下图片所示，将目标地址上的 `ether` 转到指定地址。



其中执行第二次时，就把 nonce2 地址上的 ether 恢复到我们的账户地址中了。



两次操作记录位于：

https://ropsten.etherscan.io/address/0x7ab07e42fe62b71029bc40656aa182548b936753#internaltx

https://ropsten.etherscan.io/address/0x8d8F362E422dbD39A348d783fDA0CEac12c0046f#internaltx

三、无合约隐藏 ether

用户： 0xb081ff7fe0f854db7a6e024af2aed20a1768bd97

计算出此账户地址生成的合约地址列表(生成脚本见附录)：

```
none0 = "0x71d3222234bfb7695dd54593d72197b1a5dc2f55"
none1 = "0x05395c00e0c9bb5d169481a97d287bcce23765c3"
none2 = "0x67c0c2727f37c365a2e63d1331f5d79e00e6511c"
none3 = "0xe979e4e2eb5d827bbdcaf07a78c351ff3fb5585c"
none4 = "0x441905cf7c457657e39571b0689d14f4be66506b"
none5 = "0x5ec29eeb0884ce63ba95318a177c3df7af7cb09b"
none6 = "0xd20fdc6142c3d9a9f548b862ca94ba5e8c0a1b3a"
```

能过部署合约，验证了生成的合约地址与上面对应上了。

合约1： nonce5 : 0x5ec29eeb0884ce63ba95318a177c3df7af7cb09b

合约2： nonce6 : 0xd20fdc6142c3d9a9f548b862ca94ba5e8c0a1b3a

通过销毁合约便能回收 ether 了，这里便不做进一步验证了。

四、扩展思路：多层级方式隐藏 ether

此外，我们不仅可以通过 nonce 隐藏资金在垂直的模式中，还可以从合约层次横向地隐藏资金。即生成的合约能够继续生产合约。如下面这种方法，通过多层级的 nonce 控制来隐藏资金存放地址。

```
                                              nonce+3 --> contract3
                                              nonce+2
                                              nonce+1
            nonce+2 --> contract1 --> contract2 : nonce
            nonce+1
`accountA` : nonce
```

上面的操作可以表达为 `up,up, in, in, up,up,up, in,`，或者以二进制的形式表达：`11001110` 或 `0xCE` 。意思是，先通过 `accountA` ，执行交易，在 nonce+2 处创建生产合约 `contract1` ，然后使用 `contract1` 创建 `contract2` ，执行几次交易后再创建 `contract3` 。当然，你要能够最终生成你的存放地址并回收 ether。

五、寻找丢失的 ether

找回丢失的 ether 的条件是，你转错的地址是你能创建的，并且是未来的一个nonce。

这样的场景在于，你钱包账号地址在测试环境下测试时的 nonce 比较大时生成的合约地址，如 nonce 140 时地址为 T，然后你在真实环境下，误把 ether 转到了这个地址T，此时，你的 nonce 还不到 140 ，你就可以在 nonce 140 处生成一个能销毁或有回收功能的合约，生成的这个合约的地址就等于 T 了。此时就能寻回丢失的 ether。

六、相关练习

ethernaut 的第18题`Recovery`便考察了这一点，这里就不做详细描述。

ethernaut 地址： https://ethernaut.zeppelin.solutions/

解题方法可以参考 Ali0th 大佬的 writeup： https://blog.riskivy.com/%E6%99%BA%E8%83%BD%E5%90%88%E7%BA%A6ctf%EF%BC%9Aethernaut-writeup-part-4/

七、附录：地址计算脚本

下面是两个计算此值的脚本。

python 版本：

```
# need install pyethereum module
import rlp
from ethereum import utils

address = 0x1230000000000000000000000000000000000000
nonce = 10

rlp_res = rlp.encode([address,nonce])
print(rlp_res)
sha3_res = utils.mk_contract_address(address,nonce)
print(sha3_res)
sha3_res_de = utils.decode_addr(sha3_res)
print("contract_address: " + sha3_res_de)
```

nodejs 版本：

```
//node version: v9.10.0
//module versions:
//rlp@2.0.0
//keccak@1.4.0

const rlp = require('rlp');
const keccak = require('keccak');
```

```
var nonce = 0x64; //The nonce must be a hex literal!
var sender = '0x1230000000000000000000000000000000000000'; //Requires a hex string as
input!

var input_arr = [ sender, nonce ];
var rlp_encoded = rlp.encode(input_arr);
var contract_address_long = keccak('keccak256').update(rlp_encoded).digest('hex');
var contract_address = contract_address_long.substring(24); //Trim the first 24 characters.
console.log("contract_address: " + contract_address);
```

八、资料

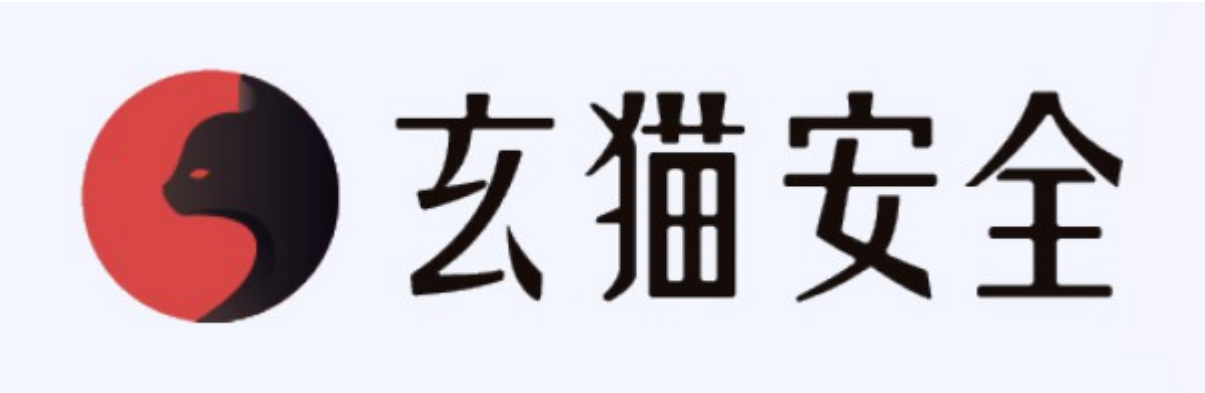
KeylessEther： https://github.com/sigp/solidity-security-blog#keyless-ether

hiding in plain sight : http://swende.se/blog/Ethereum_quirks_and_vulns.html

RLP： https://github.com/ethereum/wiki/wiki/%5B%E4%B8%AD%E6%96%87%5D-RLP

How we sent ETH to the wrong address and successfully recovered them: https://medium.com/bitclave/how-we-sent-eth-to-the-wrong-address-and-successfully-recovered-them-2fc18e09d8f6

关于玄猫安全



玄猫区块链安全实验室专注区块链安全领域，致力于提供区块链行业最专业的安全解决方案，团队成员来自于百度、阿里、360等国际顶尖安全团队，已为数十家交易所、电子钱包、智能合约等提供基础安全建设、渗透测试、漏洞挖掘、应急响应等安全服务。

玄猫安全实验室提供专业权威的智能合约审计服务、区块链专项应用评估、区块链平台安全评估等多项服务。

商务合作：Lyon.chen@xuanmao.org