Безопасность ICO контрактов (5)

Александр Половьян
alex@ledgers.world

Ether

- Ценность внутри Ethereum
- Источники появления:
    - пре-аллокация
      (размещение на счетах в genesis блоке)
    - награда майнерам
- Варианты использования:
    - покупка gas для оплаты транзакций
    - перевод на счета внутри Ethereum
      (в том числе, покупка других криптовалют/фиатных денег)

Перевод ETH

- ETH могут владеть люди и смарт-контракты
- Действия с ETH (перевод):
  - Человек — подписывает транзакции своим приватным ключом
  - Смарт-контракт – управляет своим счетом на основании поступивших транзакций. Транзакции инициируются человеком и подписываются его приватным ключом

Детали

- Потеря ключа = потеря доступа к кошельку
- Отменить транзакцию невозможно

- Перевод на несуществующий адрес – не может случиться, так как все адреса являются валидными получателями

Перевод Eth в Solidity

		Активирует fallbak	Доступный gas	on failure
addr.transfer(x)		Да	2 300	throws
add.send(x)		Да	2 300	`false`
addr.m.value(x)()	Добавляет *x ether* в вызов метода *m*	Нет	Весь *gasleft()*	`false`
майнинг (beneficiary)	beneficiary адрес майнер выставляет произвольно	Нет	Ø	NA
selfdestruct(addr)	Удаление смарт-контракта, весь gas со счета перечисляется на 0xaddr	Нет	Ø	NA

Токен

- Единица ценности созданная организацией для взаимодействия с клиентами

- Правила обращения регулируются смарт-контрактом

Особенности

- Прозрачность
- Стоимость
- Гибкость
- Отсутствие регулирования
- Мода

- Техническая сложность
- Отсутствие регулирования

Применение

- Привлечение средств (ICOs)
- Спекуляции
- Внутренний учет
- …

Популярные токены

- Tether (USDT) – идея выпустить токен который будет соответствовать 1USD
- EOS – токены для обмена на EOS coin
  в отдельном чейне
- Bancor – внутренняя валюта для обменника
  нет свободного обращения
- IOTA – использование токена не описано в WP

Компилятор

- solc
  для рабочих проектов

- solcjs
  для скриптов на node

Флаги

- `solc --help`
- `… --combined-json="abi,bin"`
- `… --allow-paths  …`
- `~$ solc —combined-json="abi,bin,bin-runtime" --allow-paths ./libs/* contracts/MyICO.sol > compiled.json`

Как сделать удобно?

```
> solc --optimize --combined-json "abi,bin" Contract*.sol >
compiled.json

...

> echo -n 'compiled = ' > compiled.js

...

> cat compiled.json >> compiled.js

...

> echo -n ';' >> compiled.js

...

> geth --preload compiled.js attach ipc:geth.ipc
```

Как дальше с этим работать?

```
> Object.keys(compiled.contracts)
…

> var abi =
JSON.parse(compiled.contracts[“myContract”].abi)
…

> var bin =
JSON.parse(compiled.contracts[“myContract”].data
…

> var deployTx = eth.contract(abi).new(…,{data:bin, …})
…
```

Настоящий production

- http://truffleframework.com
- Компиляция
- Деплоймент
- Тестирование (!!!)

Доступ из приложения

- web3py
- web3js
    - in-browser Ð-app
    - callbacks / promises / async-await
- …

Криптография с открытым ключом

- Каждый пользователь имеет 2 ключа:
  - private key для подписи сообщений
  - public key для проверки подписи

- Ключ — это последовательность символов
  Ценность приватного ключа в том, что его не знает никто кроме отправителя
- Подпись сообщения приватным ключом – это еще одна случайная строка
- Публичный ключ — производная приватного ключа, позволяет проверить подпись

Криптография с открытым ключом (формальный подход)

- Отправитель
    - m — сообщение
      p_r — приватный ключ
    - f: p_r -> p_u; получение публичного ключа
        - распространение p_u среди корреспондентов
    - g: m, p_r -> s; подпись сообщения m приватным ключом
    - отправляем сообщение m и подпись s
- Получатель
    - h: m, s, p_u -> {true, false}; проверка подлинности подписи
    - то есть, h будет равна *true*, если сообщение m было зашифровано приватным ключом, соответствующим публичному ключу p_u с результатом s

Подписи не только для транзакций

- web3.personal.sign

- ecrecover