# ЛЕКЦИЯ №2 ВВЕДЕНИЕ В БЛОКЧЕЙН

Москва, 2021

## Ethereum

Ethereum: конечный смарт-контракт и децентрализованная платформа приложений

- «Белая книга» обсуждает необходимость большего программного контроля над транзакциями
- Желание создать «децентрализованные автономные корпорации» (DAC)
- Внедряет идею «умных контрактов» как субъекта, который может отправлять и получать валюту, помимо людей

# Ethereum

**Сеть «Ethereum»**

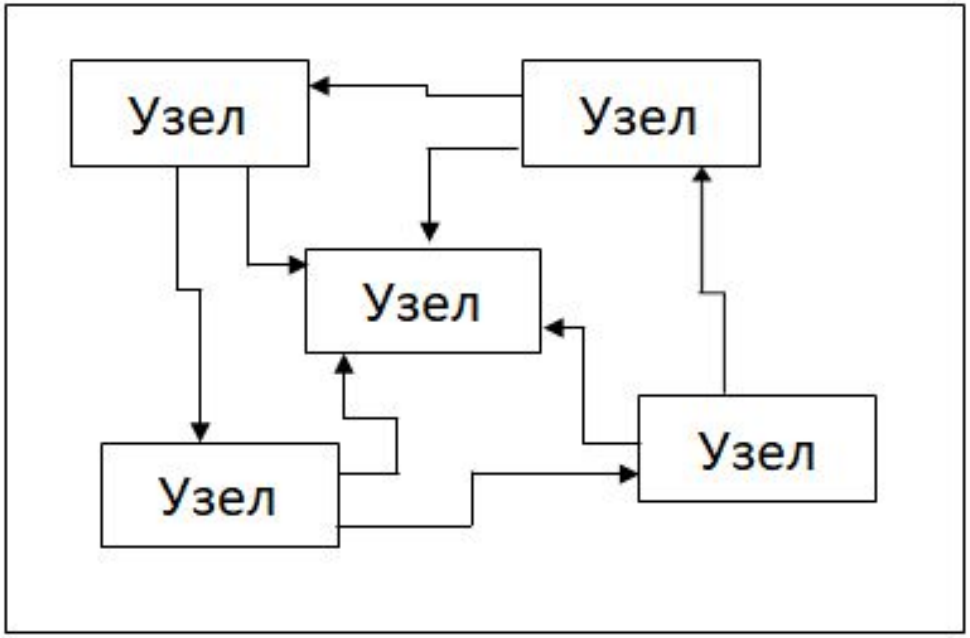
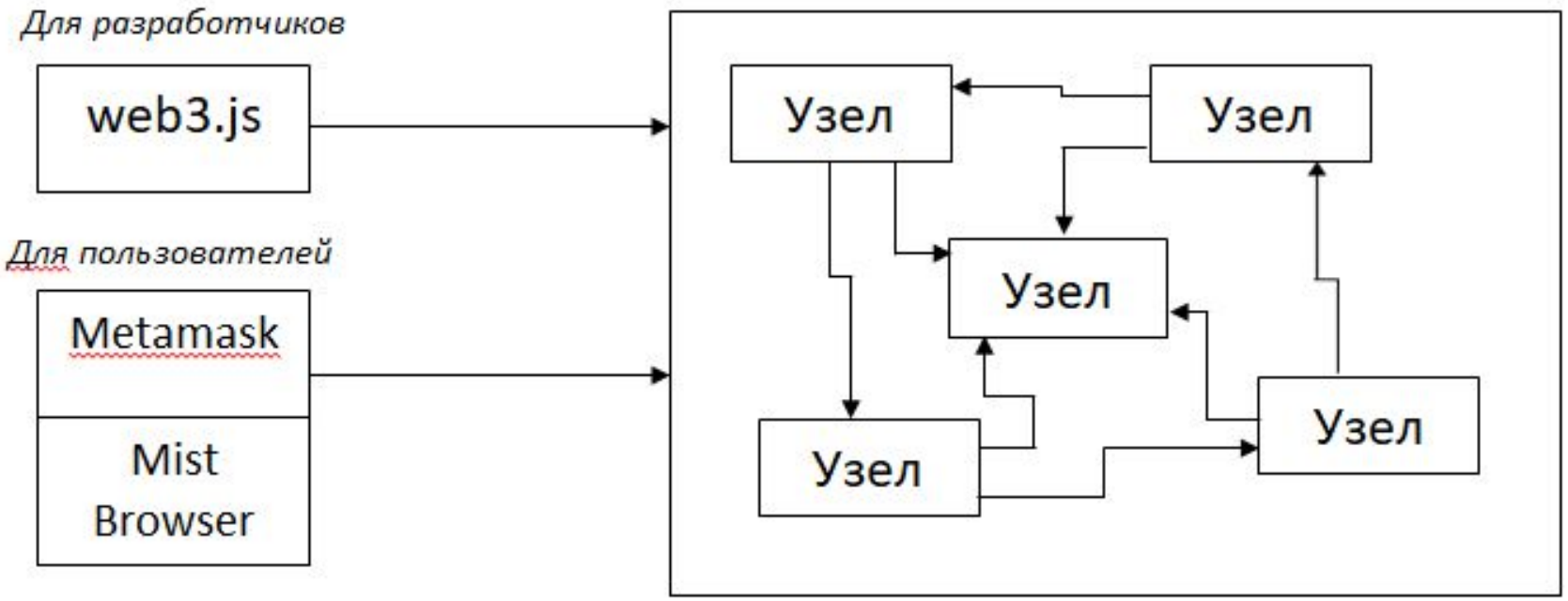


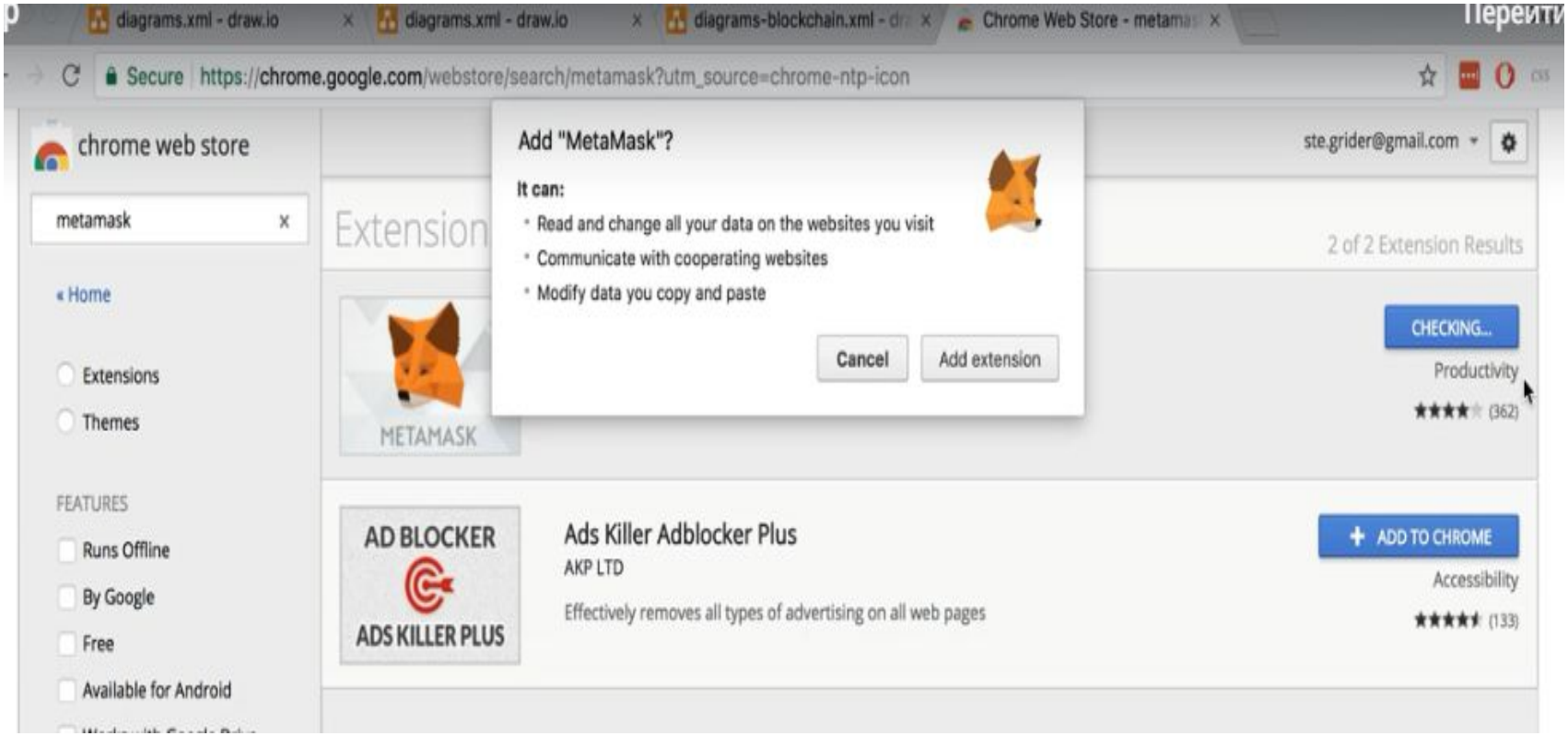

Рисунок 3

| |
|---|
| Сеть Ethereum используется для перевода денег и хранения данных |
| Есть много разных сетей Ethereum. |
| Сети образованы одним или несколькими узлами. |
| Каждый узел — это машина, на которой работает клиент Ethereum. |
| Любой может запустить узел. |
| Каждый узел может содержать полную копию блокчейна |
| Блокчейн - это база данных, которая отслеживает записи каждой транзакции, которая когда-либо происходила |

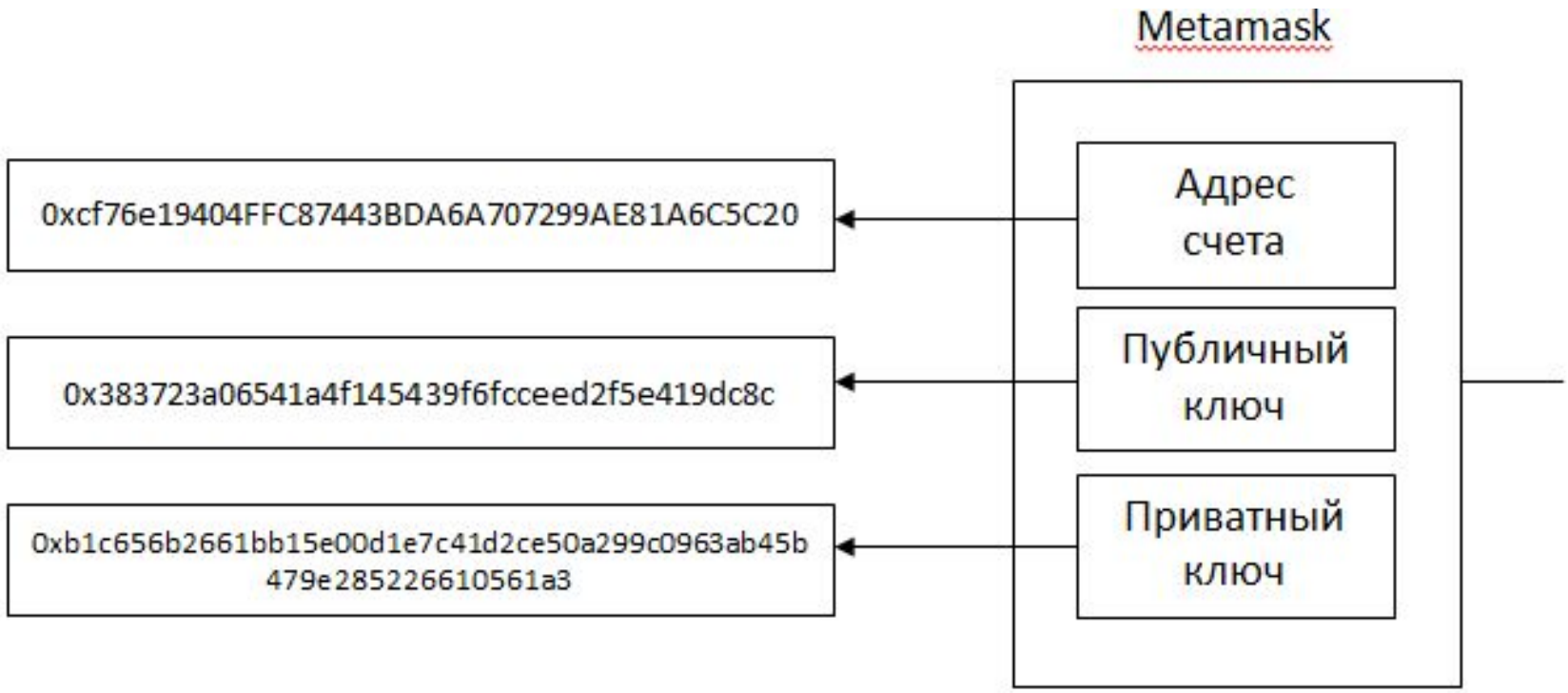
# Ethereum


Для разработчиков
web3.js
Для пользователей
Metamask
Mist
Browser
Узел
Узел
Узел
Узел
Узел

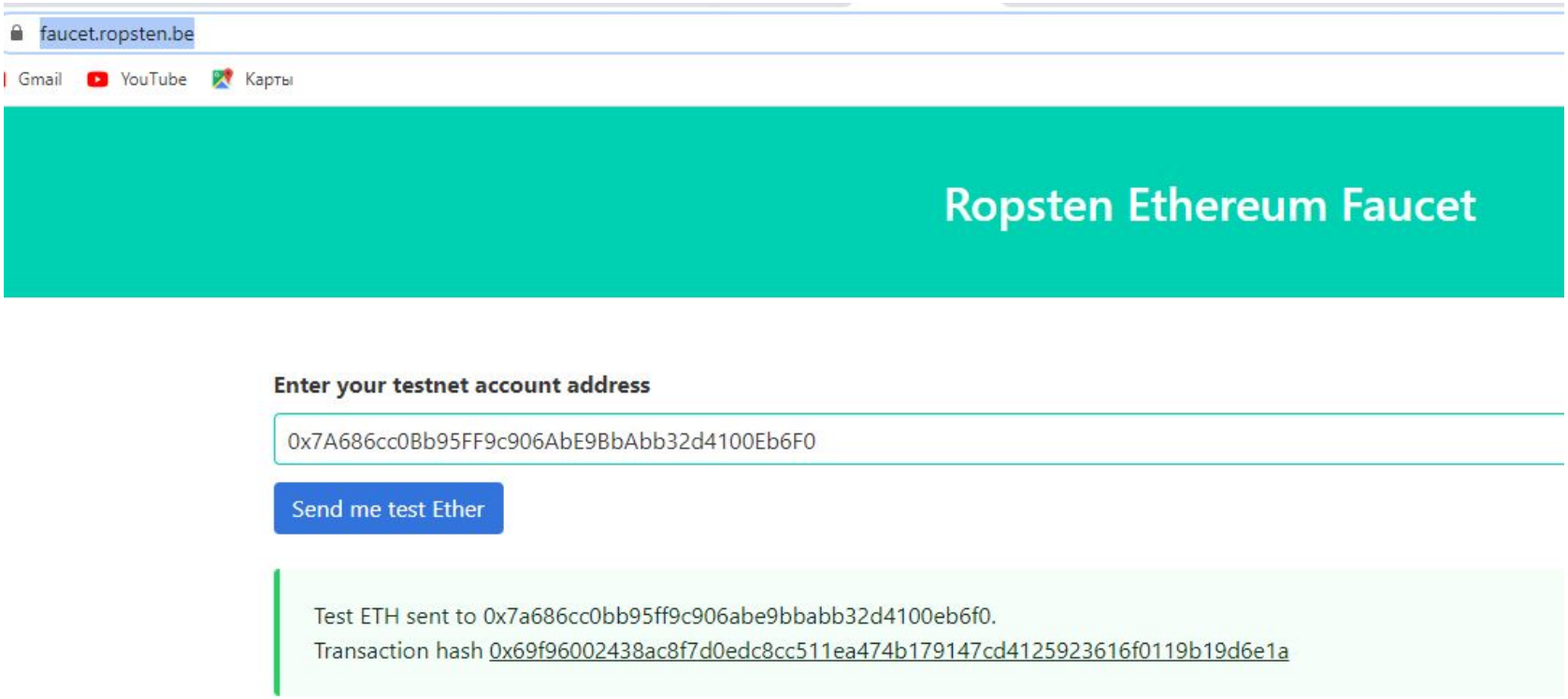
# Установка метамаска



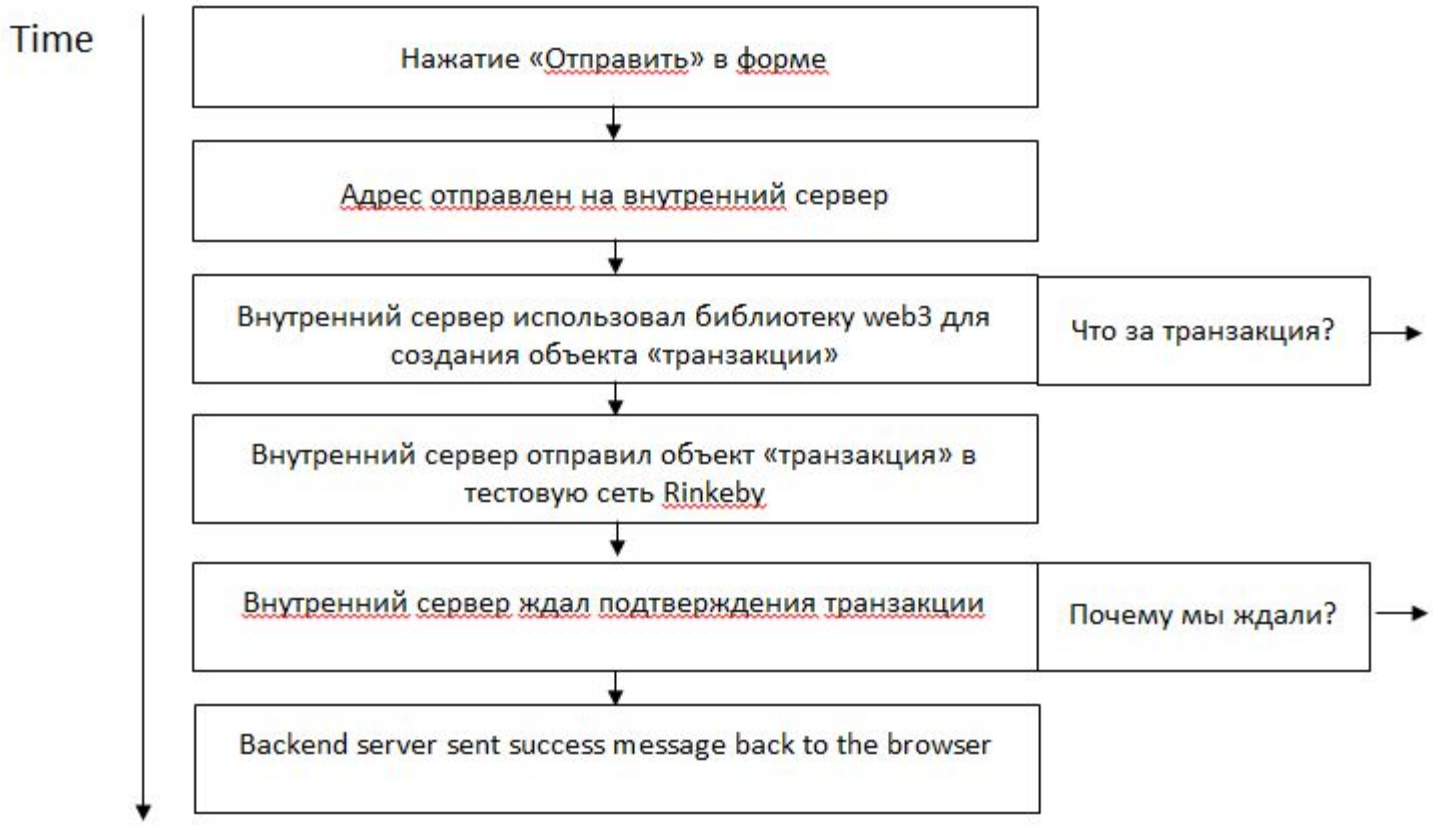
# Адрес счёта в сети



# Добавление тестовых денег



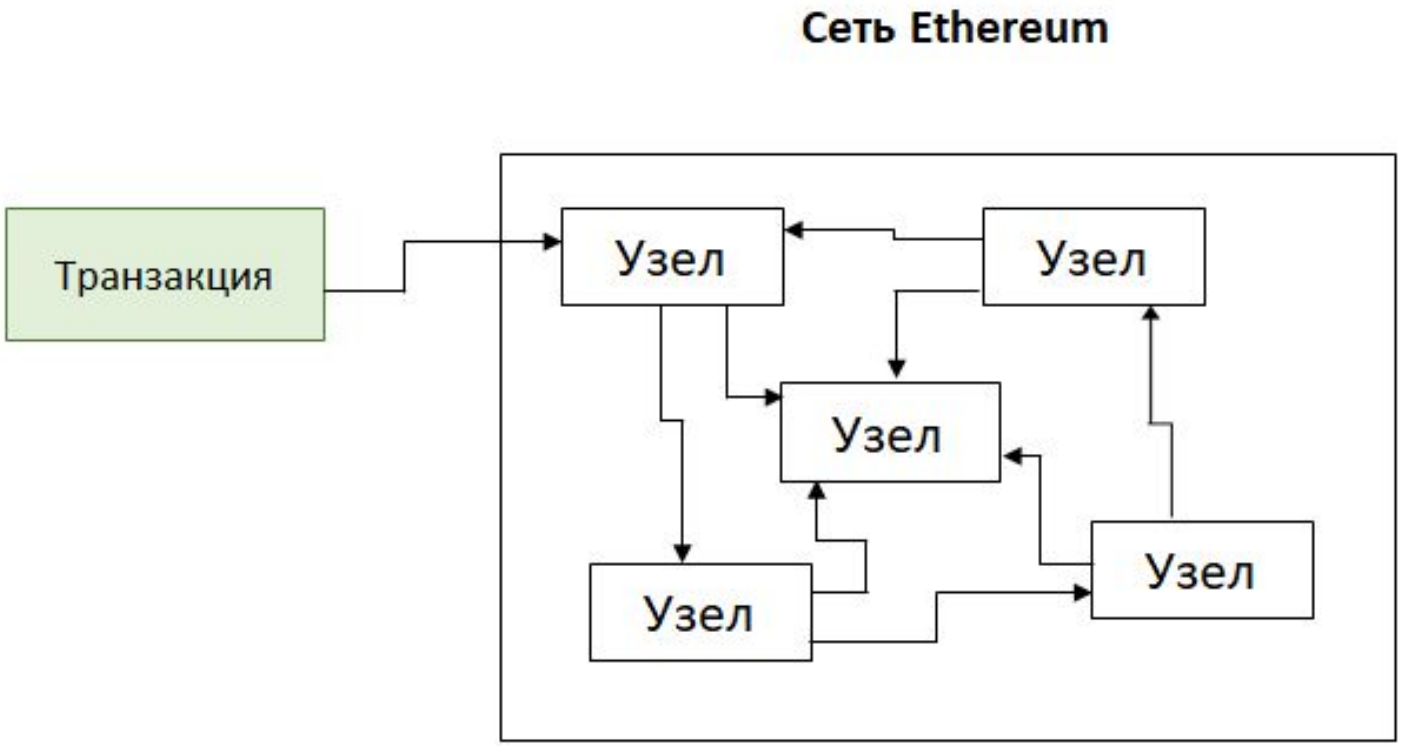
# Транзакции


Time
Нажатие «Отправить» в форме
Адрес отправлен на внутренний сервер
Внутренний сервер использовал библиотеку web3 для создания объекта «транзакции»
Что за транзакция?
Внутренний сервер отправил объект «транзакция» в тестовую сеть Rinkeby
Внутренний сервер ждал подтверждения транзакции
Почему мы ждали?
Backend server sent success message back to the browser

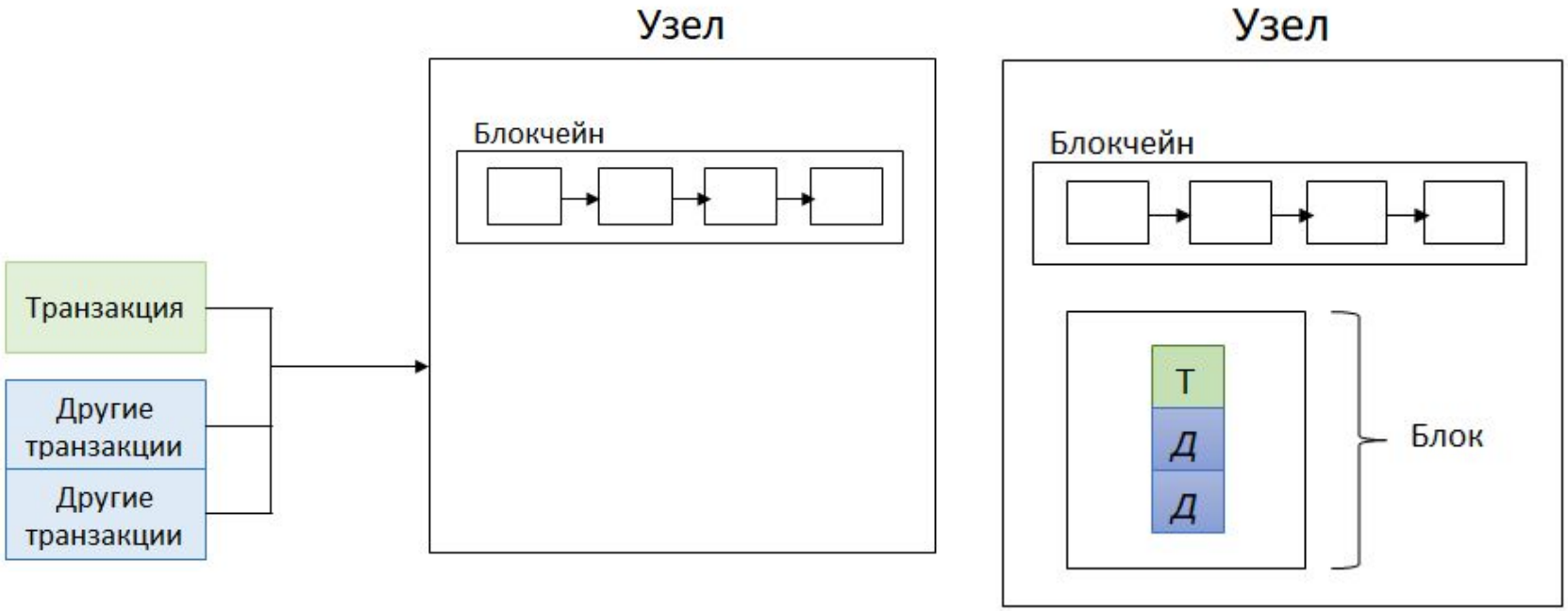
# Транзакции

| | |
|---|---|
| nonce | Сколько раз отправитель проводил транзакцию |
| to | Адрес счета, на который идут эти деньги |
| value | Количество эфира, отправляемого на указанный адрес |
| gasPrice | Количество эфира, которое отправитель готов заплатить за единицу газа, чтобы обработать эту транзакцию |
| startGas/gasLimit | Предельное количество газа, разрешенное для расходования на выполнение транзакции |
| v | Криптографические фрагменты данных, которые можно использовать для генерации адреса счета отправителя.<br>Сгенерировано из закрытого ключа отправителя. |
| r | |
| s | |

# Транзакции



# Транзакции


Узел
Блокчейн
Транзакция
Другие транзакции
Другие транзакции
Узел
Блокчейн
Т
Д
Д
Блок

# Алгоритм hashcash

**Шаг 1.** Клиент сообщает серверу, что хочет получить некоторую услугу.

**Шаг 2.** Сервер генерирует случайное слово $s$ и высылает его клиенту вместе с целым числом $k$. (В реально применяемых протоколах длина двоичной записи $s$ несколько сотен бит, а величина $k$ — несколько десятков.)

**Шаг 3** («работа» клиента). Клиент находит такое слово $x$, для которого

$$h(s||x)\,|_1^k = 00\ldots0,$$

где $w\,|_1^k$ — $k$ первых символов слова $w$, $a||b$ — конкатенация слов $a$, $b$ и 00...0 — слово, состоящее из $k$ нулей.

**Шаг 4.** Клиент высылает серверу слово $x$. Сервер вычисляет величину $h(s||x)$ и, если $k$ первых символов этого слова нули, то предоставляет клиенту требуемые услуги, иначе — не предоставляет.

## Доказательство выполнения работы

Изучим свойства этого протокола, начав с рассмотрения основного, третьего, шага. Главный вопрос — как найти слово $x$, удовлетворяющее Оказывается, что для «идеальной» хеш-функции нет лучшего подхода, чем последовательное вычисление значений $h(s||x_1)$, $h(s||x_2)$, $h(s||x_3)$, ..., где $x_1$, $x_2$, $x_3$, ... — генерируемые случайно слова (на практике это часто просто последовательные числа 1, 2, 3, ...).

Интерактивный и не интерактивный режим работы

# Доказательство выполнения

Во-вторых, мы описали так называемый интерактивный протокол, когда сервер и клиент обмениваются несколькими сообщениями. Его можно легко превратить в неинтерактивный. Для этого сервер может потребовать от клиента выполнения шага 2, т.е. поручить ему генерировать и слово $s$ (конечно, пара $s||x$ может порождаться как одно слово $u$). Клиент, нашедший такое слово, может отправлять его серверу для получения услуги; предварительный обмен сообщениями не требуется.

Здесь, однако, возникают две легкопреодолимые трудности. Клиент может попробовать использовать найденное им слово $u$ несколько раз. Для защиты от этого серверу достаточно хранить список ранее полученных слов $u$ и при получении от какого-либо клиента нового $u$ предоставлять услугу только при отсутствии $u$ в списке ранее полученных.

Другая, сходная трудность — клиент может попытаться использовать слово $u$, найденное для одного сервера, для получения услуги от другого. Для решения этой проблемы сервер может потребовать от клиентов присылать ему слова вида servername$||u$ , для которых, по-прежнему, $k$ начальных символов в $h$(servername$||u$) равны 0.

**Шаг 3*** («работа» клиента). Клиент находит такое слово $x$, для которого

$$h(s||x) \leq \Omega, \tag{9.5}$$

где целое число $\Omega$ — параметр метода, для которого должны выполняться условия $0 \leq \Omega \leq 2^{\alpha} - 1$. Понятно, что при случайном выборе слова $s||x$ вероятность выполнения неравенства (9.5) равна $\Omega/2^{\alpha}$ и, следовательно, при увеличении $\Omega$ на единицу эта вероятность увеличивается на $1/2^{\alpha}$. При $\alpha = 256$ или 512 эта величина очень мала, поэтому возможно плавное уменьшение $\Omega$, и увеличение обратной величины.

Рассмотрим некоторое сообщество, участники которого время от времени производят документы $x_1, x_2, \ldots$, представленные в виде цифровых файлов.

## Алгоритм датирования

Рассмотрим некоторое сообщество, участники которого время от времени производят документы $x_1, x_2, \ldots$, представленные в виде цифровых файлов.

При подписании текста TSA задействует секретный ключ. Открытый ключ TSA должен быть достоверно известен всем участникам сообщества, и с его помощью каждый участник может проверить подлинность подписи TSA.) Обозначим подпись TSA к документу $u$ через $\Sigma_{\mathrm{TSA}}(u)$, тогда подписанный документ будет выглядеть как $u||\Sigma_{\mathrm{TSA}}(u)$.

Опишем теперь протокол датирования документа. Пусть один из участников, Алиса, создала документ $x$ и хочет добавить к нему сведения о дате. Опишем работу протокола по шагам:

## Алгоритм датирования

**Шаг 1.** Алиса вычисляет хеш-функцию $h(x)$ и отправляет ее к TSA.

**Шаг 2.** TSA, получив $h(x)$, формирует слово (файл) $h(x)||t_x$, где $t_x$ — время получения документа $x$ (скажем, в заранее обусловленном формате: ГОД, МЕСЯЦ, ДЕНЬ, ЧАС, МИНУТА, СЕКУНДА).

**Шаг 3.** TSA подписывает слово $h(x)||t_x$, т.е. вычисляет подпись $\Sigma_{\text{TSA}}(h(x)||t_x)$, и высылает ее и $t_x$ Алисе.

**Шаг 4.** Алиса, получив время и подпись TSA, формирует документ с «заверенной» датой $w(x) = x||t_x||y$, где $y = \Sigma_{\text{TSA}}(h(x)||t_x)$.

Любой участник, скажем Боб, может проверить подлинность даты в некотором документе $w^* = x^*||t_x^*||y^*$. Для этого Боб должен вычислить $h(x^*)$ и проверить подлинность подписанного сообщения $h(x^*)||t_x^*||y^*$, исходя из того что $y^* = \Sigma_{\text{TSA}}(h(x^*)||t_x^*)$.

Если это сообщение выдерживает проверку, то Боб делает вывод о том, что действительно документ $x^*$ был датирован в момент $t_x^*$; в противном случае Боб делает вывод о том, что $w^*$ подделан, т.е. $x^*$ был создан после даты $t_x^*$.