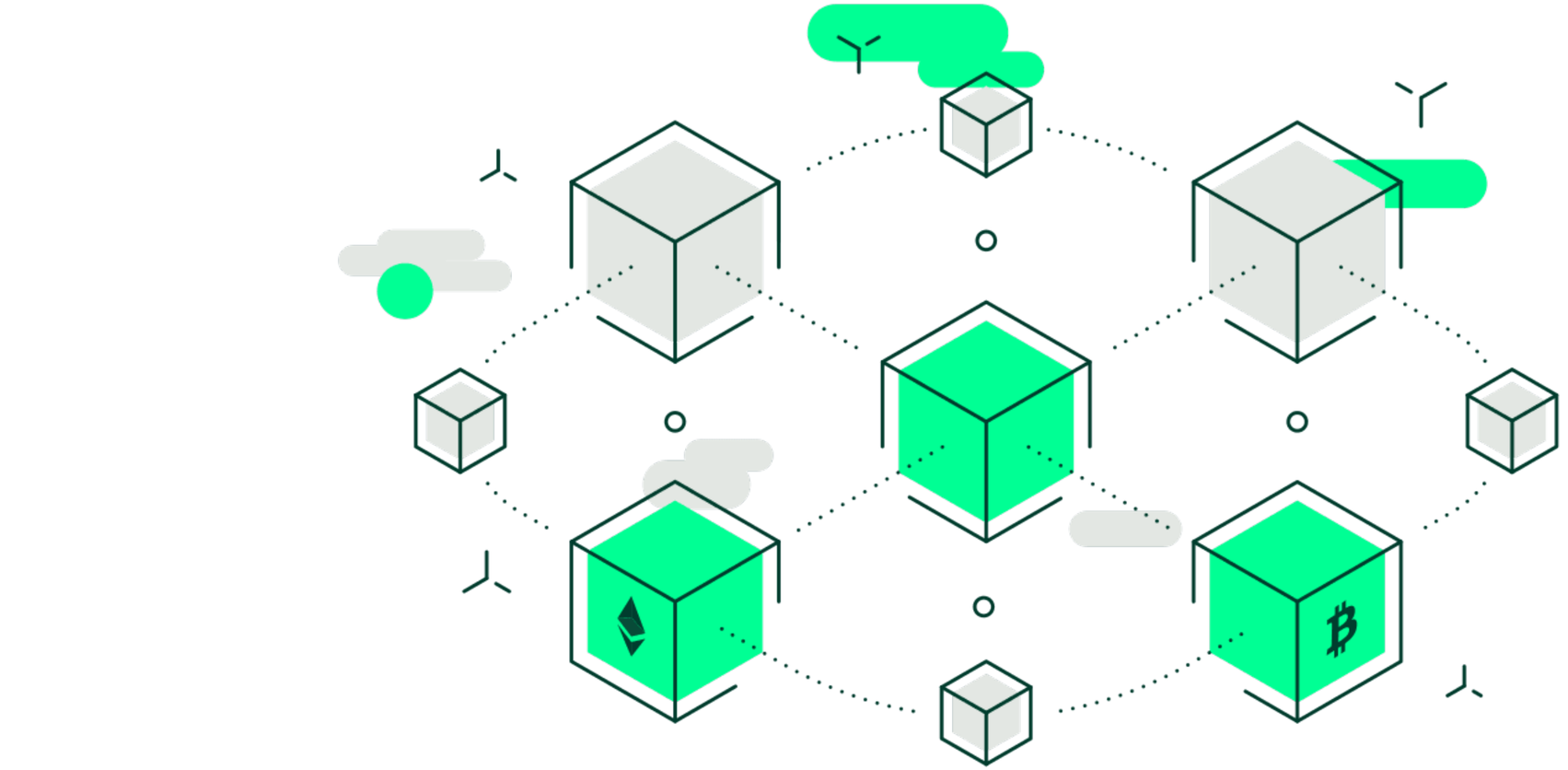
JavaScript и способы взаимодействия с блокчейном на примере Ethereum

Андрей Макаров

Что такое этот ваш блокчейн?

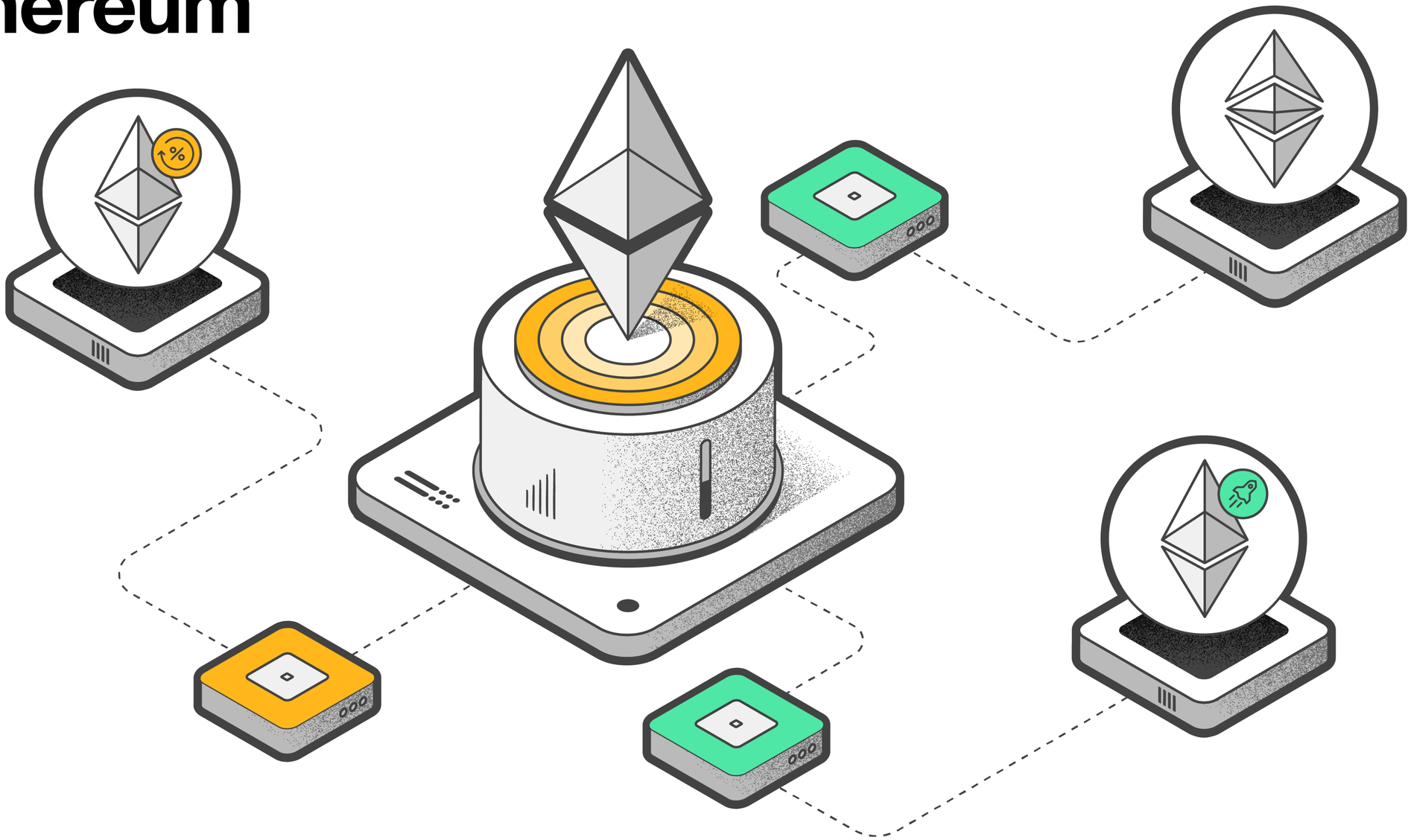
- Хэш
- Цепочка блоков
- Децентрализация
- Консенсус



Базовые принципы блокчейна

- Симметричное шифрование
- Асимметричное шифрование
- Хэширование
- Электронная подпись
- Децентрализация

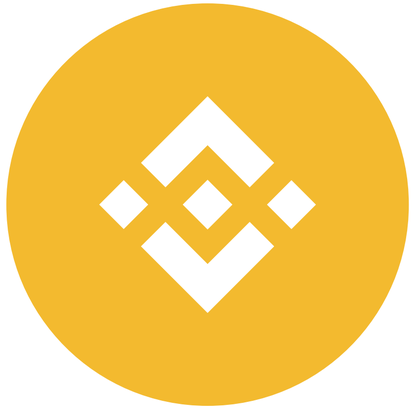
Ethereum



Ether

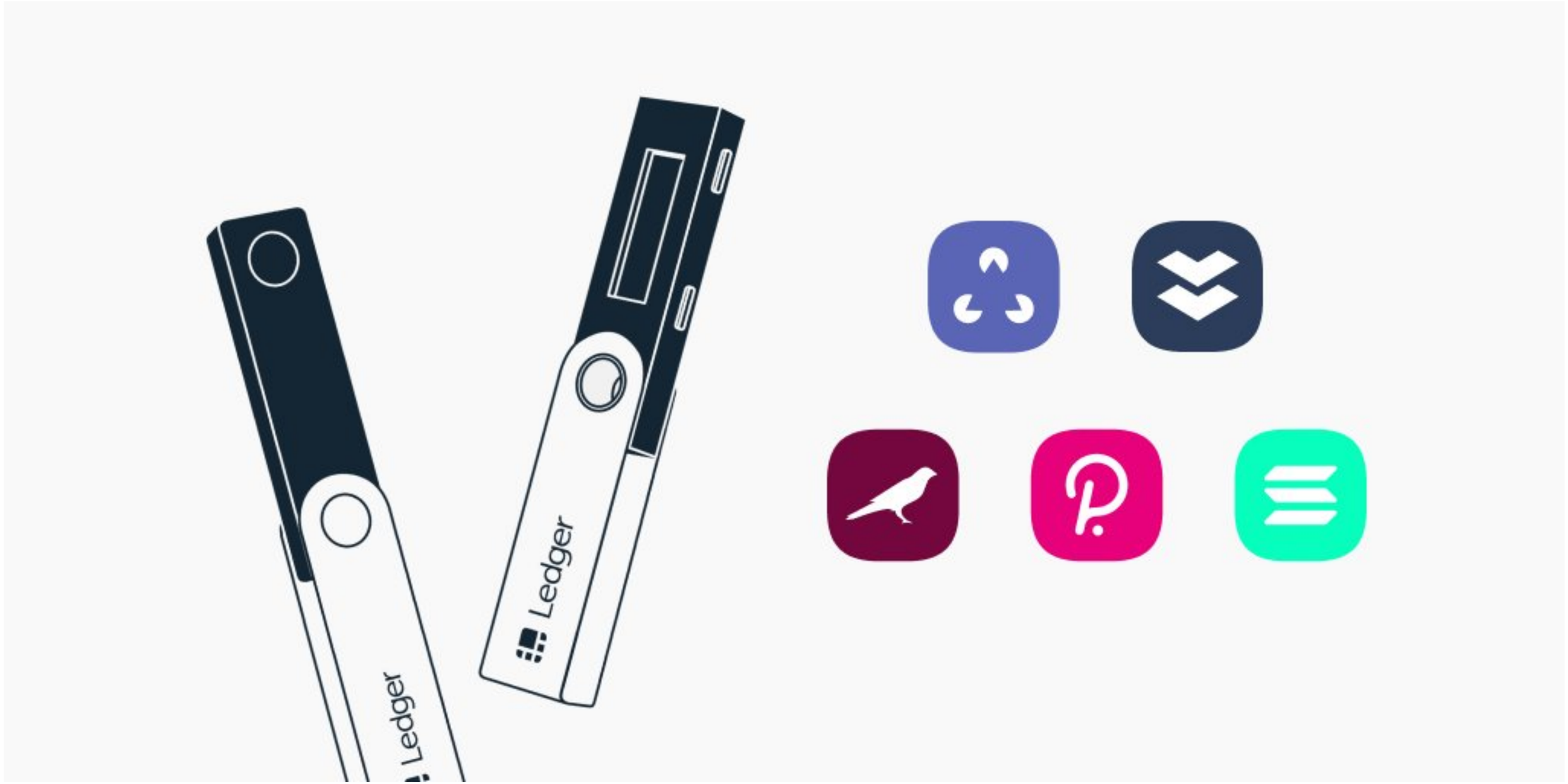
Смарт контракты

Кастодиальные кошельки

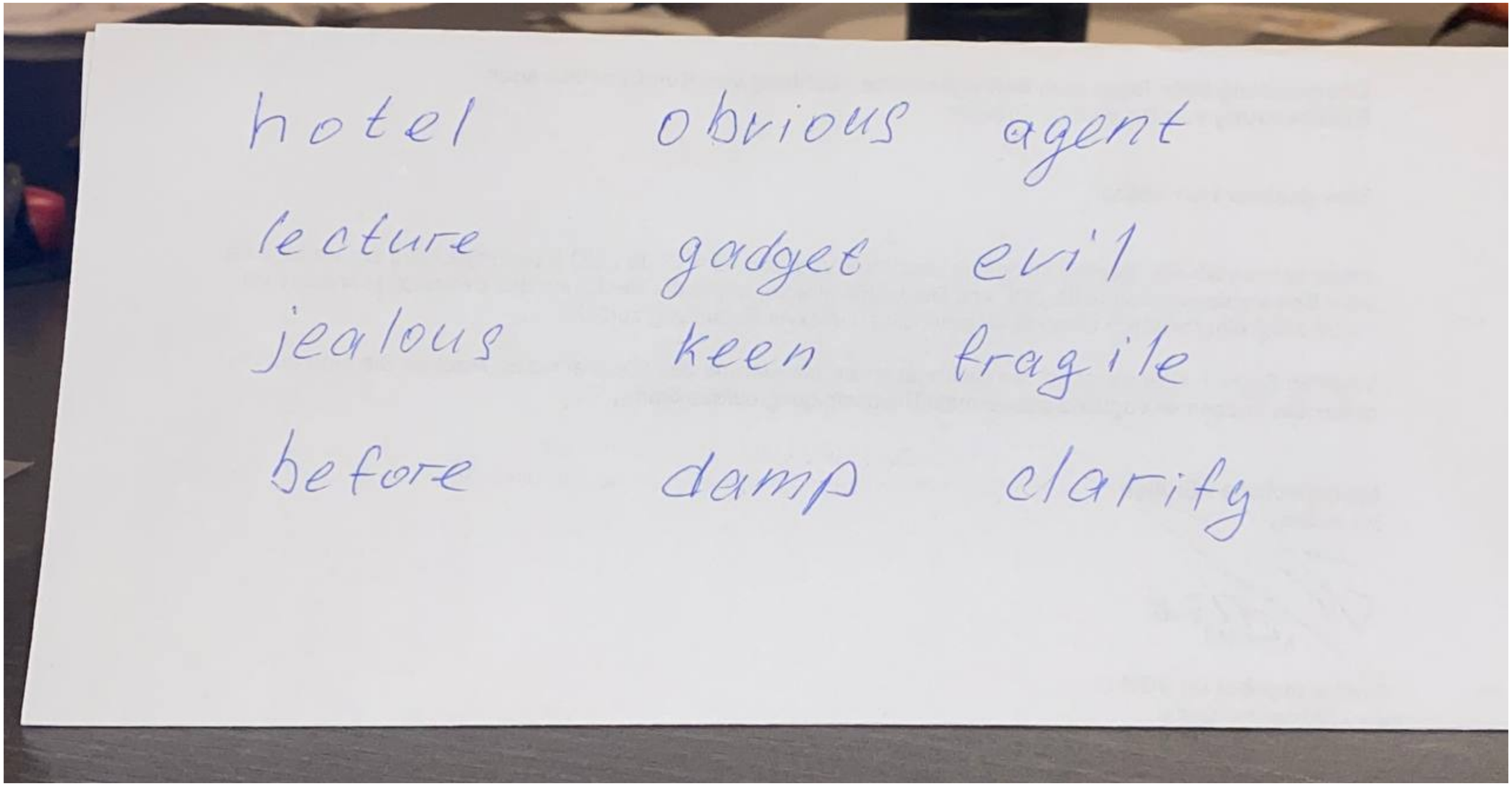


Горячие кошельки (программные)

Аппаратные кошельки



Бумажные =)


hotel
lecture
jealous
before
obvious
gadget
keen
damp
agent
evil
fragile
clarify

Что должен уметь делать кошелек?

- Хранение приватной информации
- Безопасность
- Резервное копирование и восстановление
- Взаимодействие с экосистемой
  - Отображение информации о количестве активов
  - Отправка активов
  - Подпись сообщений и смарт контракты

Обзор основных библиотек для Web3

- Web3.js
- Ethers.js

Отправка транзакции при помощи ethers.js

```
1.import {ethers} from 'ethers'
2.
3.async function sendTransaction(to: string, amount: number) {
4.  const provider = new ethers.providers.StaticJsonRpcProvider('http://localhost:8545', {
5.    chainId: 1337,
6.  })
7.
8.  const signer = new
  ethers.Wallet('0x69b39aa2fb86c7172d77d4b87b459ed7643c1e4b052536561e08d7d25592b373');
9.
10. const balance = await provider.getBalance(signer.getAddress())
11. console.log('before:', balance.toString())
12.
13. const tx = { to, value: ethers.utils.parseEther(amount) }
14. const signedTx = await signer.signTransaction(tx);
15. const txHash = await provider.sendTransaction(signedTx)
16. console.log('txHash', txHash)
17.
18. const balanceNew = await provider.getBalance(signer.getAddress())
19. console.log('after', balanceNew.toString())
20.}
21.
22. sendTransaction('0xA16842b28FF96Ec695008996F0D85BE705A2c4Dd', 10).then(() => { })
```

Генерация мнемонической фразы

```
1.  function chunkSubstr(str: string, size: number) {
2.    const numChunks = Math.ceil(str.length / size)
3.    return Array.from({length: numChunks}, (_, i) => str.substring(i * size, (i + 1) * size))
4.  }
5.
6.  function bytesToBits(bytes: Buffer): string {
7.      return Array.prototype.slice.call(bytes, 0).map(b => (b).toString(2).padStart(8, "0")).join("");
8.  }
9.
10.  function deriveChecksumBits(bytes: Buffer): string {
11.     let hash = Buffer.from(sha256(bytes), 'hex');
12.     return bytesToBits(hash).substring(0, bytes.length * 8 / 32);
13.  }
14.
15.  export function entropyToMnemonic(entropy: Buffer) {
16.     let bits = entropy.reduce((acc, byte) => acc + byte.toString(2).padStart(8, "0"), "") +
   deriveChecksumBits(entropy)
17.     let chunks = chunkSubstr(bits, 11).map((binary) => parseInt(binary, 2));
18.     return chunks.map((binary) => mnemonicWordlist[binary]).join(' ');
19.  }
```

Получение сид фразы

```
1.  import {pbkdf2Sync} from "pbkdf2";
2.
3.  export function getSeedPhrase(mnemonic: string, passphrase = ''): Buffer {
4.    let saltBytes = `mnemonic${passphrase}`
5.    return pbkdf2Sync(mnemonic, saltBytes, 2048, 64, 'sha512');
6.  }
```

Получение приватного ключа

- HD (Hierarchical Deterministic) path
- m’/44’/60’/0’/0

```
1. import HDKey from 'hdkey';
2.
3. const hdkey = HDKey.fromMasterSeed(seed);
4. const privateKey =  hdkey.derive(hdPath).privateKey.toString('hex');
```

Получение публичного ключа и адреса

```
1.  import {keccak256} from "js-sha3";
2.
3.  import elliptic from 'elliptic';
4.
5.  export function accountInfo(privateKey: string): { publicKey: Buffer, address: string } {
6.    const ec = new elliptic.ec('secp256k1');
7.    const key = ec.keyFromPrivate(Uint8Array.from(Buffer.from(privateKey, 'hex')));
8.    const pk = key.getPublic(false, 'array').slice(1);
9.
10.    const publicKeyHash = keccak256(Buffer.from(pk));
11.    return {
12.        publicKey: Buffer.from(key.getPublic(true, 'array')),
13.        address: '0x' + publicKeyHash.slice(-40),
14.    }
15. }
16.
```

Взаимодействие с блокчейном через JSON-RPC

```
1.  import {EVM_ENDPOINT} from "@/eth/constants";
2.
3.  export async function makeJsonRPCRequest<T>(method: string, params: any[] = []): Promise<T> {
4.     const resp = await fetch(EVM_ENDPOINT, {
5.       method: 'POST',
6.       headers: {
7.         'Content-Type': ‘application/json',
8.       },
9.       body: JSON.stringify({
10.         jsonrpc: '2.0',
11.         method,
12.         params,
13.         id: 1,
14.       })
15.     });
16.
17.     const json = await resp.json();
18.     return json.result;
19. }
```

Запрос баланса

```
import BN from "bn.js";

try {
 const balance = await makeJsonRPCRequest<string>('eth_getBalance', [address, 'latest'])
 console.log('before:', balance)

} catch (e) {
  console.log('error:', e.message())
}
```

Подготовка транзакции к отправке

```ts
import {UnsignedTransaction} from "@ethersproject/transactions";
import {makeJsonRPCRequest} from "@/eth/make-json-rpc-request";
import BN from "bn.js";
import {EVM_CHAIN_ID, MIN_GAS} from "@/eth/constants";

export async function prepareTransaction(from: string, to: string, amount: BN): Promise<UnsignedTransaction> {
    const nonce = await makeJsonRPCRequest<string>('eth_getTransactionCount', [from, 'latest']);
    const gasPrice = await makeJsonRPCRequest<string>('eth_gasPrice');

    let estimateGas = MIN_GAS;
    try {
        const resp = await makeJsonRPCRequest<string>('eth_estimateGas', [{
            from,
            to,
            value: '0x' + amount.toString('hex'),
            maxFeePerGas: gasPrice,
            maxPriorityFeePerGas: gasPrice,
        }]);

        estimateGas = (new BN(resp, 16)).mul(new BN(2));
    } catch (e) {
    }

    return {
        chainId: EVM_CHAIN_ID,
        data: '0x',
        gasLimit: '0x' + BN.max(estimateGas, MIN_GAS).toString('hex'),
        type: 2,
        maxFeePerGas: gasPrice,
        maxPriorityFeePerGas: gasPrice,
        nonce: parseInt(nonce, 16),
        to: to,
        value: '0x' + amount.toString('hex'),
    };
}
```

Подпись и отправка транзакции

```
import BN from "bn.js";
import {makeJsonRPCRequest} from "@/eth/make-json-rpc-request";
import elliptic from "elliptic";
import {keccak256} from "js-sha3";
import {serialize} from "@ethersproject/transactions";
import {prepareTransaction} from "@/eth/prepare-transaction";

export async function transfer(privateKey: string, from: string, to: string, amount: BN): Promise<string> {
    const tx = await prepareTransaction(from, to, amount);
    const encodedTx = serialize(tx);

    const txHash = keccak256(Buffer.from(encodedTx.slice(2), 'hex'));

    const ec = new elliptic.ec('secp256k1');

    const signature = ec.sign(
        Buffer.from(txHash, 'hex'),
        Buffer.from(privateKey, 'hex'),
        {canonical: true}
    );

    const signedTx = serialize(tx, {
        v: 27 + (signature.recoveryParam ? signature.recoveryParam : 0),
        r: '0x' + signature.r.toString('hex'),
        s: '0x' + signature.s.toString('hex'),
    })

    return await makeJsonRPCRequest<string>('eth_sendRawTransaction', [signedTx]);
}
```

Шифрование приватного ключа keccak256

```
1.  import BN from 'bn.js';
2.  import {CURVE_N} from './constants';
3.  import {accountInfo} from './account-info';
4.  import {keccak256} from "js-sha3";

6.  function hashPasswordToBN(message: string) {
7.    return new BN(keccak256(message), 'hex');
8.  }
9.
10. export function encryptPrivateKey(privateKey: string, password: string): { nonce: string,
  publicKey: string } {
11.   const hash = hashPasswordToBN(password);
12.   const nonce = new BN(privateKey, 'hex').sub(hash).umod(CURVE_N);
13.   const {publicKey} = accountInfo(privateKey);
14.   return {
15.     nonce: nonce.toString('hex'),
16.     publicKey: publicKey.toString('hex'),
17.   };
18. }
19.
```

Дешифрование приватного ключа keccak256

```
1. import BN from 'bn.js';
2. import {CURVE_N} from './constants';
3. import {accountInfo} from './account-info';
4. import {keccak256} from "js-sha3";
5.
6. function hashPasswordToBN(message: string) {
7.   return new BN(keccak256(message), 'hex');
8. }
9.
10. export function decryptPrivateKey(shareEncrypted: { nonce: string, publicKey: string }, password:
  string): string {
11.   const userInputHash = hashPasswordToBN(password);
12.   const pk = new BN(shareEncrypted.nonce,
  ‘hex').add(userInputHash).umod(CURVE_N).toString('hex').padStart(64, '0');
13.   const info = accountInfo(pk);
14.   if (info.publicKey.toString('hex') !== shareEncrypted.publicKey) {
15.     throw new Error('Incorrect password');
16.   }
17.
18.   return pk;
19. }
20.
```

Угрозы для кошелька

- Утеря приватного ключа / мнемоника
- Хранение в памяти незашифрованного приватного ключа/мнемоника
- Фишинговый сайт
- Человеческий фактор

Защита

- Дайте пользователю сохранить свой мнемоник и проверьте что он это точно сделал!
- Обучение пользователя грамотности
- Зашифровать приватный ключ / мнемоник фразу
- Использовать холодные кошельки, такие как Ledger
- Использовать сторонние хранилища, такие как Metamask или Wallet

Локальные сети для тестирования web3 приложения

- npx hardhat init
- npx hardhat node

Started HTTP and WebSocket JSON-RPC server at http://127.0.0.1:8545/

Accounts

========

WARNING: These accounts, and their private keys, are publicly known.

Any funds sent to them on Mainnet or any other live network WILL BE LOST.

Account #0: 0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266 (10000 ETH)

Private Key: 0xac0974bec39a17e36ba4a6b4d238ff944bacb478cbed5efcae784d7bf4f2ff80

Спасибо за внимание