Ambush Attacks

on Ethereum's (and Diem) Contract Addresses

Kostas Chalkias, 2019

Ethereum Address types

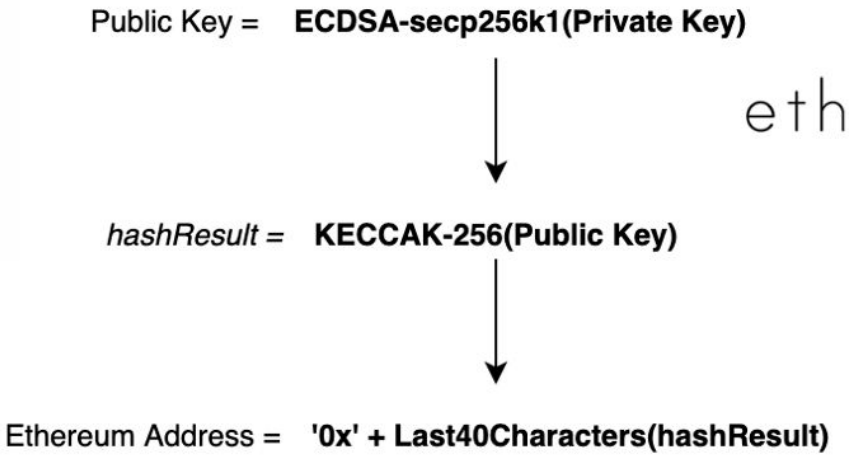
Two types of accounts:

EOA (externally owned account) : **[user address]** - controlled by private keys.

Contract Accounts: **[smart contract address]** - controlled by contract’s code.

Both of them 160bit (20 bytes) in size.

EOA Addresses



Public Key = **ECDSA-secp256k1(Private Key)**

hashResult = **KECCAK-256(Public Key)**

Ethereum Address = **'0x' + Last40Characters(hashResult)**

Private key: 32 bytes secret x

Public key: xP

Brute-forcing PKs: xP + P = (x+1)P

Contract Addresses

Ethereum smart contracts can be created both by other contracts (using Solidity’s `new` keyword) and by regular EOA accounts.

Since Feb 2019 there exist 2 algorithms (opcodes) to create a smart contract address. CREATE1 and the most recent CREATE2.

CREATE1

account’s seq_id

```
keccak256(deployingAddr + nonce).last20B()
```

CREATE2

user selected

```
keccak256(0xff + deployingAddr + salt + keccak256(bytecode)).last20B()
```

The main benefit of CREATE2

The whole idea behind CREATE2 is to **make the resulting address independent of future events**.

Regardless of what may happen on the blockchain, it will always be possible to deploy the contract at the precomputed address.

***Counterfactual** instantiation is a concept that gained popularity in the context of generalized state channels. It refers to the creation of a contract that could happen, but has not; yet the fact that it could is enough.*

Bits of Security in Cryptography

According to major security/cryptography agencies, anything below 128bits of security is considered vulnerable and exploitable. See **keylength.com**

We have evidence that anything close to 2^96 is (theoretically) breakable TODAY:

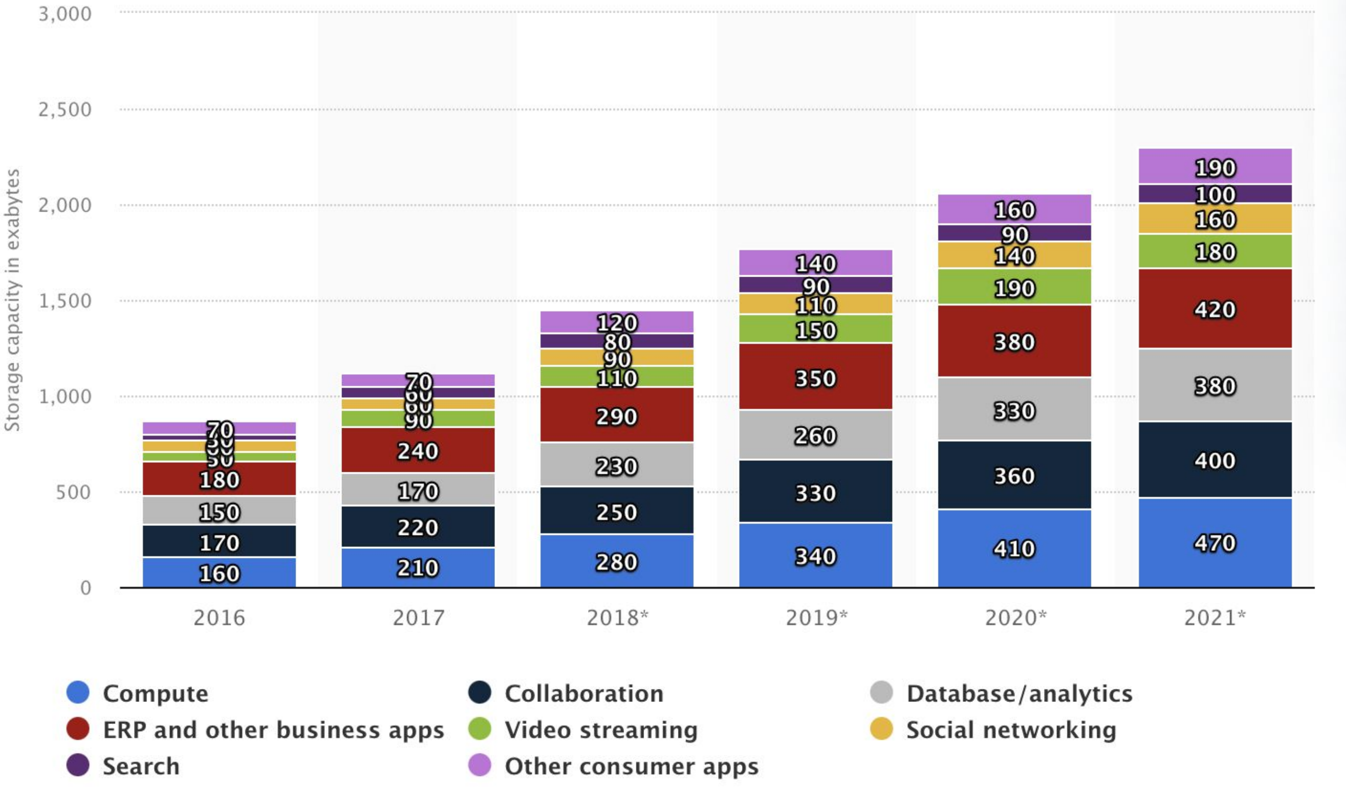
Example:
Bitcoin hashrate hit 180 Exahash per sec (Feb 2021), roughly 2^68 hps

1 year secs ~= 2^25
1 year hps = 2^(68+25) = 2^93

Data center storage capacity worldwide from 2016 to 2021

Data center storage:
2^72 bytes

Est. world storage:
~2^75 bytes



© Statista 20

Hash Function attacks

pre-image: given y, find a x, such that $h(x) = y$

Effort: 2^256 brute-force in sha256

2nd pre-image: given x, find a $x' \neq x$, such that $h(x) = h(x')$

Effort: 2^256 brute-force in sha256

multi-target pre-image: given a list of y_i, find a x, such that $h(x) = \text{any } y_i$

Effort: 2^(256-k) in sha256 (2^k elements in y array)

collision: find any two distinct x, x′, such that h(x) = h(x′)

Effort: 2^128 in sha256 (due to birthday paradox)

Oops, that's half of the hash function's output bytes

Ambush Eth contract address collision attack

Because Ethereum addresses are a 20 bytes (160 bits) hash output -> it implies that a collision attack would cost 2^80 hash operations + memory / storage.

Note: there are ways to perform collision attacks without storage (similar to rainbow password attacks).

CREATE1

Pick random

Pick a reasonable int i.e. up to 1000?

```
keccak256(deployingAddr + nonce).last20B()
```

CREATE2

Pick random

Pick random

Add random padding

```
keccak256(0xff + deployingAddr + salt + keccak256(bytecode)).last20B()
```

Objective: Find 2 contracts with the same address

1. Write 2 contracts: (a) the one you want to deploy and attack and (b) a “smart” dummy one
2. Execute a collision attack until both addresses match (i.e., pick random salts)
3. Publish the 1st contract (i.e., a DeFi, ERC20, L2 Rollup etc)
4. Wait until the contract receives enough locked funds (you have to make it popular)
5. When you are ready for the attack, publish the dummy contract.
   It now overrides the previous address.
6. Drain the balance(s) in the 1st smart contract & transfer them to your controlled account.

Objective: Contract to EOA address collision

1. Write the contract you want to deploy and attack
2. Pick a random private key x and create public key xP
3. Execute a collision attack until both addresses match (i.e., pick random salts and do (x+k)P)
4. Publish the contract (i.e., a DeFi, ERC20, L2 Rollup etc)
5. Wait until the contract receives enough locked funds (you have to make it popular)
6. When you are ready for the attack, send a transaction with the EOA address
7. Drain the balance(s) in the 1st smart contract & transfer them to your controlled account.

Update (2021): https://eips.ethereum.org/EIPS/eip-3607

Other side effects

1. An attacker can convince a user to send funds to an account before it is deployed. Some applications require this behaviour (e.g. state channels).

2. A chain reorg can happen after a contract is deployed. If the reorg removes the contract deployment transaction the funds can still be accessed using the private key.

3. A contract can self-destruct, with the stated intention that ERC20s (or other tokens) in the contract would be burned. However, they can now be accessed by a key for that address.