One of the more complicated pre-requisites ofstate expiry is the need to add more address space to hold new “address periods” every year. The main available solution to this is address space extension, which increases address length to 32 bytes. However, this requires complicated logic for backwards compatibility, and even then existing contracts would have to update.

One alternative is address space compression

. We decide on some yet-unused 4-byte prefix, and first agree that it is no longer a valid address prefix. Then, we use those addresses for address periods in the state expiry proposal; the 20 byte address could be formatted as follows:

[4 byte prefix] [2 byte address period] [14 byte hash]

This greatly reduces the backwards compatibility work required. The only backwards compatibility issue would be dealing the possibility that some existing application has a future CREATE or CREATE2 child address that collides with the 4 byte prefix (really, 5 bytes of collision would be needed to be a problem as address periods would not reach 2 bytes for ~256 years).

However, reducing the hash length to 14 bytes comes with a major sacrifice: we no longer have collision resistance

.

The cost of finding two distinct privkeys or initcode (or one with a privkey and one with initcode) that have the same address decreases to 2^{56}

, which is very feasible for a well-resourced actor. Brute force attacks, finding a new privkey or initcode for an existing

address, would still take 2^{112}

time and be computationally infeasible for an extremely long time.

It’s also worth noting that even without address space extension, collisions today take 2^{80}

time to compute, and that length of time is already

within range of nation-state-level actors willing to spend huge amounts of resources

. For reference, the Bitcoin network performs 2^{80}

SHA256 hashes once every two hours

.

This raises the question: what would actually break if we no longer have collision resistance?

Contracts already on-chain: SAFE

If a contract is already on-chain, it is safe (assuming the contract cannot self-destruct orSELFDESTRUCT is banned entirely). In fact, many other blockchains simply give contracts sequentially assigned

addresses (usually <= 5 bytes). The only reason Ethereum cannot do this for all accounts is that we need to somehow

handle users being able to receive funds when they interact with Ethereum for the first time, and do not yet have an on-chain account.

Externally owned accounts (EOAs): SAFE

A user would be able to generate two privkeys that map to the same address. However, there’s nothing that a user could actually do

with this ability. Anyone wanting to send funds to that user already trusts that user.

Not-yet-created single-user smart contract wallets: SAFE

Most modern smart contract wallets (including theERC 4337 account abstraction proposal) support deterministic addresses. That is, a user with a private key could compute the address of their smart contract wallet locally, and give this address to others to send funds to them. ERC 4337 even allows ETH sent to the address to be used to pay for the fees of the contract deployment and the first UserOperation from the address.

A user could

generate two different private keys with the same ERC 4337 address, or even two private keys where one has X as its ERC 4337 address and the other has X as its EOA address. However, once again there’s nothing that a user could do with this

ability.

Not-yet-created multi-user wallets: NOT SAFE

Suppose that Alice, Bob and Charlie are making a 2-of-3 ERC 4337 multisig wallet, and plan to receive funds before they publish it on chain. Suppose Charlie is evil and wishes to steal those funds. Charlie can do the following:

- Wait for Alice and Bob to provide their addresses A and B
- Grind a collision, finding C1

and C2

that satisfy get_address(A, B, C1) = get_address(C2, C2+1, C2+2)

- Provide C1 as their address

Once the address get_address(A, B, C1)

receives funds, they can deploy the initcode with {C2, C2+1, C2+2}

and claim the funds for themselves.

The easiest solution is to deploy the address before it receives funds. A more involved solution is to use a value (eg. a future blockhash, a VDF output) that is not known at the time A, B and C are provided but becomes known soon after as a salt that goes into address generation. An even more involved solution is to MPC the address generation process.

Complex state channels: NOT SAFE

Suppose Alice and Bob with addresses A and B are playing chess inside a channel. A typical construction is that they both sent funds into an on-chain channel contract, and they pre-signed a 2-of-2 message that authorizes the channel contract to send those funds into a (not-yet-created) chess contract that instantiates A and B as the two players, requires them to play a chess game with transactions, and sends the funds to the winner.

The problem, of course, is that Bob could grind a collision get_chess_contract_address(A, B1) = get_chess_contract_address(B2, B2+1)

, and extract the funds.

A natural solution is to revert to something like the way state channels worked before CREATE2 was an option: the 2-of-2 transfer would specify a hash, and there would be a single central resolver that would only transfer funds to a contract that had code that hashed to that particular hash. Alternatively, a state channel system could require that all contracts must be pre-published ahead of time; the chess contract would have to be written to support multiple instances

of the game.

General principles of a post-collision-resistance world

In a world where addresses are no longer collision-resistant, the easiest rule for developers and users to remember would be: if you send coins or assets or assign permissions to a not-yet-on-chain address, you are trusting whoever sent you that address

. If your intention was to send funds to that person, you are fine. If they are giving you an address to what they claim is a not-yet-published address with a given piece of code that they don't have a backdoor to, remember that the same address could also be publishable with a different

piece of code that they do

have a backdoor to.

In such a world, applications would generally just be designed in a way where all addresses except for first-time user wallets are already on chain.