Here are some common categories of contracts that exist currently on mainnet or will soon exist, and an attempt to analyze how they fare under the introduction of storage maintenance fees ("rent") in their present form, and how they could be modified to survive the adoption thereof. This list is intended to be useful for evaluating both concrete proposals for storage maintenance fees and proposals for high-level languages that target blockchains with storage maintenance fees.

Anyone is welcome to suggest more examples!

Tokens (ERC20)

Effect of rent

- vulnerable to griefing attacks: anyone can send 1 wei (or equivalent) to an unlimited number of accounts, causing the token contract to incur permanent ongoing costs.

Solution I

- store the balance associated with each user in a contract made with CREATE2 using the user's address as a seed. Require the user to keep these contracts topped up. Note that for efficiency this can be combined across many applications by creating a generic UserStorageFactory

contract, where a contract C can, with permission of user U, add an entry at a contract with a CREATE2 address made with sha3(C, U) as a seed.

Solution II

- store a Merkle root of the balances, and require the user to provide a Merkle branch to spend coins. To mitigate "collisions" (Merkle branches failing due to 2 people trying to spend at the same time), the contract can also store a history of the last, say, 100 Merkle tree nodes added/removed, and can combine that data with branches based on old data to generate valid branches (see The Stateless Client Concept for more discussion).

Cryptokitties

Effect of rent

- vulnerable to griefing attacks: anyone can keep breeding kitties, which continually add to the contract's ongoing storage expenditure.

Solution

- every kitty becomes a separate contract, owner responsible for upkeep.

Multisig wallets

Effect of rent

- in many present designs, in a M of N multisig wallet, any one of the N participants can create an unlimited number of deposit requests, and there is no way to delete these requests. The Ethereum Foundation wallet seems to be not

vulnerable to this, because 4 of 7 participants can simply keep revoking transactions. However, the Gnosis multisig (code here). seems to be vulnerable.

Solution

- store each pending transaction as a contract, with a short TTL (as there's no particular reason for long TTLs in any case)

Stateless multisig wallets

Effect of rent

- some classes of multisig wallets do not store state except for a list of owners and a sequence number (eg this Vyper multisig). These are not vulnerable to attacks because they have O(1) storage.

ENS

Effect of rent

- vulnerable to griefing attacks: anyone can register an unlimited number of domains, and the _entries

mapping in the code will store all of their addresses forever.

Solution

- because ENS is contract-based already, this is easier than the others: just switch to CREATE2-based addresses, using the domain as a seed, allowing the contract address for any given domain to be generated in real-time.

On-chain order books

Effect of rent

- vulnerable to griefing attacks: anyone can make an unlimited number of orders with arbitrarily unfavorable terms, and these orders will stay in the state forever (eg. see Oasisdex code)

Solution I

- open orders become contracts.

Solution II

- orders are stored in a Merkle priority queue; accepting an order involves simply providing Merkle proofs of the pop

operation. Note that this also solves the on-chain sorting problem.

Smart contracts representing agreements

Effect of rent

- some implementations of smart contracts representing agreements (eg. crowdfunds, escrows, CFDs…) for efficiency reasons put all agreements into the same contract (eg. see this very old implementation of crowdfunds in Serpent). These would be vulnerable to griefing attacks. However, designs where a separate contract is used for each agreement are not

vulnerable to griefing attacks.

Solution

- switch to a design where a separate contract is used for each agreement.

Privacy-preserving contracts (mixers, anonymous voting, etc)

Effect of rent

- some classes of privacy-preserving mixers work using the following mechanism. Anyone can join the mechanism by providing 1 coin along with a commitment c

. They can then withdraw the coin at any time by providing a value I

such that f1(x) = c

and f2(x) = y

for some f1

and f2

where x

is a secret. Because of cryptographic machinery, there is only one possible I

for each c

, and using x

it's possible to prove that some given I

corresponds to one

of the commitments that has been published, without revealing which one. I

values need to be stored so that they cannot be double-spent. This theoretically includes ring signatures and many kinds of ZK-SNARK schemes. With storage maintenance fees, each I

value stored leads to an ongoing expense on chain, leading to griefing vulnerabilities.

Solution I

- put commitments into “buckets”, each bucket corresponding to eg. the c

values submitted within one specific month, and for each I

value submitted require the prover to make a proof based on a specific bucket. Store the bucket ID. Run a background process that deletes I

values that are too old.

Solution II

- put commitments and I values into a Merkle tree.

Solution III

- put used I values into an accumulator which allows succinct proofs of non-membership, eg. a STARK-based accumulator if one wants a purely hash-based construction.