Setting the scene

We already have techniques to support confidential currency transfer.

How can we bring confidentiality to smart contracts?

I’d argue this problem is far from solved (even in theory) as it isn’t clear how to support multi-user encrypted inputs without the use of trusted managers.

Why is privacy for smart contracts harder than for currency transfers?

- May involve complex operations with more than just addition. Conditions to be satisfied (on the inputs) depend on the particular application. Contracts may take in encrypted inputs belonging to different users.
- Concurrency issues in account model from using ZKPs.
- Efficiency issues from using ZKPs compounded further in trying to support more general computation.

Barring the use of trusted hardware or trusted managers, we can classify the two major approaches to private smart contracts as follows:

1. Homomorphic encryption-based (e.g. Zether)
2. ZKP-based (e.g. Zkay, Zexe)

Elaborating on the two approaches

We have named these approaches based on which

cryptographic primitive allows for performing the private computation itself. Note that both use ZKPs but to different ends.

1. Homomorphic encryption (HE)-based (idea: use homomorphic properties of encryption scheme to perform private computation

)

- User provides encrypted inputs and ZKP (proving application-specific conditions have been satisfied).
- Miners check ZKP and perform computation directly on encrypted inputs.
- Computations supported depend on the homomorphic properties of the chosen encryption scheme. For example, if an additively homomorphic encryption scheme is chosen then only addition can be performed on the encrypted inputs.

- User provides encrypted inputs and ZKP (proving application-specific conditions have been satisfied).
- Miners check ZKP and perform computation directly on encrypted inputs.
- Computations supported depend on the homomorphic properties of the chosen encryption scheme. For example, if an additively homomorphic encryption scheme is chosen then only addition can be performed on the encrypted inputs.

- ZKP-based (idea: use ZKP to prove offline computation was done correctly

)

- User performs computation on her plaintext inputs.
- User encrypts outputs and produces ZKP showing this computation/update was done correctly.
- Miners check ZKP.

- User performs computation on her plaintext inputs.
- User encrypts outputs and produces ZKP showing this computation/update was done correctly.
- Miners check ZKP.

Considerations

- Minimizing user’s vs. miner’s work (HE-based approach minimizers user’s work whereas ZKP-based approach minimizes miner’s work). In HE-based approach, the encryption scheme must be chosen carefully with performance in mind.

- Computational requirements on user (are we expecting the user to have access to powerful machines?)
- Neither approach supports multi-user encrypted inputs out of the box. We could consider more sophisticated cryptographic primitives to support this functionality via MPC or multi-key FHE. Unfortunately, neither of these are particularly efficient.

Our work

Idea: Extend Approach #1

(HE-based) using FHE

We choose an FHE scheme modeling computation as arithmetic circuits (e.g. BGV, BFV) + ZKPs (short discrete log proofs, bulletproofs).

Considerations

- Choosing a performant FHE scheme to minimize miner's time performing homomorphic computations
- Minimizing user's computational requirements (i.e. expecting the user to have access to only a modest machine)
- Ciphertext growth from homomorphic multiplication
- Proofs for lattice-based relations can be expensive
- Possibility as a layer 2

Finally, a link to our work!