Zero-Knowledge Credentials for Smart Contracts

Lucas Switzer, Tjaden Hess

May 12, 2018

Abstract

Public blockchains present unique opportunities for the implementation of autonomous and trustless systems, but suffer from trade-offs between privacy and expressivity. In this paper we present an implementation of a zkSNARK-based anonymous credential scheme for the Ethereum blockchain and give benchmarks for usage costs. We present as well an example application.

1 Introduction

While blockchains have found use cases in publicly accessible distributed systems, they pose a challenge in that due to their public nature it is currently impossible to attest to aspects of one's identity without some trusted credential issuer.

1.1 Prior Work

A generalized scheme was proposed by Garman et al. [1]

2 Properties and Features

3 Implementation

3.1 dApp Framework and Credential-Enabled Contracts

The properly interface with the ZKID Service a dApp will have to provide an annotated ABI for its dependant contract(s). The required annotations describe the required credentials for a given method within the smart contract and will be used to propery construct public inputs and subsequently generate proofs for credentials. An annotated ABI function would appear as follow:

```
{"constant":false ,
 "name":"Join",
 "inputs":[] ,
  "outputs":[],
  "payable":false,
  "type":"function","
   "credentials":[
     {"contract_salt":"0xdeadbeef",
     "high_bound":150,
     "low_bound":18,
     "k_factor":1,
     "description": "Confirms holder is over 18."
     }
   ]
```

Note that the description field is not nessesary for generation / verification of proofs, but is supplied soley for the use of the credential-acknowledgement framework described later.

Additionally, all credential-enabled contracts are required to implement a GetMerkleTreeAddress function that can provide the caller with the IPFS address of its merkle root. This is pertinent to the generation of the nessesary Merkle Tree membership proofs and proof generation will be unsuccessful if ignored.

3.2 ZKID Service

3.2.1 libsnark

3.2.2 JSON RPC Service

The ZKID client application acts as a JSON RPC service that can handle requests from client dApps. For the simplest implementation, the RPC service offered a single procedure: GenerateProofs. This procedure takes as input all the fields nessesary to construct the required set of public inputs for the proofs. If proof generation was successful, the service will respond with a set of generated proofs as well as their relative public inputs. The returned proofs can then be sent to the verifier contract.

3.3 Verifiers

It is expected, although not required, that credential-enabled contracts perform their own verification. For this reason, we have provided a Solidity library enables contracts to verify proofs autonomously. The verifier library follows a scheme similar to that of Zokrates [insert citation here]. The library exposes a single method to be utlized by the higher-level contract: verifyTx. This method does precisely what the name implies and verifies as set of proofs encoded as bytes and returns true if all proofs where sucessfully verified and false otherwise.

It is important to note that all verification calculations are done on-chain and therefore can potentially require immense amounts of gas. For this reason, the number of proofs requested for verification should be kept to a minimum.

3.4 Issuers

Although there is no strict template for an issuer (asside from the credential contruction), we purpose a scheme that we believe will offers most of the elementary functions of an issuer.

3.4.1 InterPlanetary File System (IPFS)

As designed, the ZKID service expects verifiers to store their issued-credential merkle trees using the distributed file system service, InterPlanetary File System (IPFS). By utilizing IPFS, the ZKID service promotes the distributed nature of its environment and further encourages issuers to take advantage of said distributed nature. All issuer contracts must supply a GetMerkleTreeAddress method that returns the IPFS address of their merkle root. Every issuer must also append the hashes of every issued credential to their merkle tree so that verifiers and clients can construct merkle tree paths and subsequently merkle tree membership proofs as nessesary.

4 Benchmarks

5 Example Application

To demonstrate the facilities provided by the our system we developed a simple lottery dApp that requries participants to be both over the age of 18 and have American Citizenship to participate. Along with the dApp, we created a credential-acknowledgement framework that would allow applications to prompt users for the required credentials before any proofs are generated.

6 Future Work

7 Conclusion

References

[1] Christina Garman, Matthew Green, and Ian Miers. Decentralized Anonymous Credentials. URL `http://eprint.iacr.org/2013/622`.