Solana ZK Token Proof Program

The native Solana ZK Token proof program verifies a number of zero-knowledge proofs that are tailored to work with Pedersen commitments and ElGamal encryption over the elliptic curvecurve25519 . The program was originally designed to verify the zero-knowledge proofs that are required for theSPL Token 2022 program. However, the zero-knowledge proofs in the proof program can be used in more general contexts outside of SPL Token 2022 as well.

- Program id:ZkTokenProof1111111111111111111111111111111
- Instructions:ProofInstruction

Pedersen commitments and ElGamal encryption

The ZK Token proof program verifies zero-knowledge proofs for Pedersen commitments and ElGamal encryption, which are common cryptographic primitives that are incorporated in many existing cryptographic protocols.

ElGamal encryption is a popular instantiation of a public-key encryption scheme. An ElGamal keypair consists of an ElGamal public key and an ElGamal secret key. Messages can be encrypted under a public key to produce a ciphertext. A ciphertext can then be decrypted using a corresponding ElGamal secret key. The variant that is used in the proof program is thetwisted ElGamal encryption over the elliptic curvecurve25519 .

The Pedersen commitment scheme is a popular instantiation of a cryptographic commitment scheme. A commitment scheme allows a user to wrap a message into a commitment with a purpose of revealing the committed message later on. Like a ciphertext, the resulting commitment does not reveal any information about the containing message. At the same time, the commitment is binding in that the user cannot change the original value that is contained in a commitment.

Interested readers can refer to the following resources for a more in-depth treatment of Pedersen commitment and the (twisted) ElGamal encryption schemes.

- Notes
- on the twisted ElGamal encryption
- A technicaloverview
- of the SPL Token 2022 confidential extension
- Pretty Good Confidentialityresearch paper

The ZK Token proof program contains proof verification instructions on various zero-knowledge proofs for working with the Pedersen commitment and ElGamal encryption schemes. For example, theVerifyRangeProofU64 instruction verifies a zero-knowledge proof certifying that a Pedersen commitment contains an unsigned 64-bit number as the message. TheVerifyPubkeyValidity instruction verifies a zero-knowledge proof certifying that an ElGamal public key is a properly formed public key.

Context Data

The proof data associated with each of the ZK Token proof instructions are logically divided into two parts:

- Thecontext
- component contains the data that a zero-knowledge proof
- is certifying. For example, context component for aVerifyRangeProofU64
- instruction data is the Pedersen commitment that holds an unsigned 64-bit
- number. The context component for aVerifyPubkeyValidity
- instruction data is
- the ElGamal public key that is properly formed.
- Theproof
- component contains the actual mathematical pieces that
- certify different properties of the context data.

The ZK Token proof program processes a proof instruction in two steps:

1. Verify the zero-knowledge proof data associated with the proof instruction.
2. If specified in the instruction, the program stores the context data in a
3. dedicated context state account.

The simplest way to use a proof instruction is to execute it without producing a context state account. In this case, the proof instruction can be included as part of a larger Solana transaction that contains instructions of other Solana programs. Programs should directly access the context data from the proof instruction data and use it in its program logic.

Alternatively, a proof instruction can be executed to produce a context state account. In this case, the context data associated with a proof instruction persists even after the transaction containing the proof instruction is finished with its execution. The creation of context state accounts can be useful in settings where ZK proofs are required from PDAs or when proof data is too large to fit inside a single transaction.

Proof Instructions

The ZK Token proof program supports the following list of zero-knowledge proofs.

Proofs on ElGamal encryption

- VerifyPubkeyValidity
- :
- The ElGamal public-key validity proof instruction certifies that an ElGamal
- public-key is a properly formed public key.
- Mathematical description and proof of security{[Notes]
- VerifyZeroBalance
- :
- The zero-balance proof certifies that an ElGamal ciphertext encrypts the
- number zero.
- Mathematical description and proof of security{[Notes]

Equality proofs

- VerifyCiphertextCommitmentEquality
- :
- The ciphertext-commitment equality proof certifies that an ElGamal
- ciphertext and a Pedersen commitment encode the same message.
- Mathematical description and proof of security{[Notes]
- VerifyCiphertextCiphertextEquality
- :
- The ciphertext-ciphertext equality proof certifies that two ElGamal
- ciphertexts encrypt the same message.
- Mathematical description and proof of security{[Notes] Previous Runtime Sysvar Cluster Data Next Anatomy of a Validator