ErgoScript: A More Expressive Alternative to Bitcoin Script

Ergo Developers

{ergoplatform}@protonmail.com

Every coin in Bitcoin is protected by a program in the stack-based Bitcoin Script language. An interpreter for the language is evaluating the program against a redeeming program (in the same language) as well as a context (few variables information about the state of the blockchain system). The evaluation produces a single boolean value as a result. While Bitcoin Script allows for some contracts to be programmed, its abilities are limited since many instructions were removed after denial-of-service and security issues were discovered. To add new cryptographic primitives to the language (such as ring signatures), a hard-fork is required.

Generalizing and overcoming the limitations of Bitcoin Script, we introduce a notion of an *authentication language* where a transaction verifier is running an interpreter with three inputs: 1) a *proposition* defined in terms of the language; 2) a *context* and 3) a *proof* (defined in the *language of proofs*) generated by a prover for the proposition against the same context. The interpreter is producing a boolean value and must finish for any possible inputs within constant time.

We designed ErgoScript as a call-by-value, higher-order functional language without recursion with widely known concise Scala/Kotlin syntax. It supports single-assignment blocks, tuples, optional values, indexed collections with higher-order operations, short-cutting logicals, ternary 'if' with lazy branches. All operations are deterministic, without side effects and all values are immutable. ErgoScript in not turing-complete, however it is expressive enough to make the whole transactional model of Ergo turing complete.

ErgoScript defines a guarding proposition for a coin as a logic formula which combines predicates over a context and cryptographic statements provable via Σ-protocols with AND, OR, k-out-of-n connectives. A user willing to spend the coin first evaluates the proposition over known context and entire spending transaction yielding a *Σ-protocol statement*. Then the prover is turning the statement into a signature with the help of a Fiat-Shamir transformation. A transaction verifier (a full-node in a blockchain setting) evaluates the proposition against the context and checks the signature. Language expressiveness is defined by a set of predicates over context and a set of Σ-protocol statements. We show how the latter can be extended with a soft-fork by using versioning conventions. We propose a set of context predicates for a Bitcoin-like cryptocurrency with a guarantee of constant verification time. We provide several examples: zero knowledge ring and threshold signatures, pre-issued mining rewards, crowd-funding, demurrage currency, DEX, LETS, ICO, non-interactive CoinJoin, etc.

References

1. Sarah Meiklejohn et al. Zkpdl: A language-based system for efficient zero-knowledge proofs and electronic cash. In *USENIX Security Symposium*, volume 10, pages 193–206, 2010.