Towards a quantum-safe transaction signature in Ethereum

Authors:
S. Bistarelli
M. Fiore
I. Mercanti
M. Mongiello
Our proposal

- Interface implementation of an Ethereum node for supporting different signature methods.

- Post-quantum resistant Blockchain

- No fork needed
Background

• Blockchain systems depend heavily on cryptographic methods to ensure security, protect user privacy, and enhance system performance.

• Blockchain uses wallets with key-pairs for authentication and signing processes.
  • Private key: sign transactions
  • Public key: verify transactions

• The development of quantum computing prompts growing worries on the chain security.
State of the art

• Several related work in literature.
  • Secure cryptocurrency scheme\textsuperscript{[1]}.
  • Quantum key distribution\textsuperscript{[2]}.
  • Management of cryptographic primitives in smart contracts\textsuperscript{[3]}.

• Introducing new features and applying fixes to a Blockchain cause forks in the chain.

• Proposal: development of an interface to support different signature methods in an Ethereum client.


Post-quantum algorithms

• **Key-pair generation**: generation of a user’s private key starting from his public key.

• **Transaction signature**: generation of a user’s private key starting from the signature of a transaction.

• **Block hashes**: writing new data in a block, rebuilding the chain.
Post-quantum algorithms

• **Key-pair generation**: a post-quantum algorithm can quickly generate a user’s private key starting from his public key.

• **Transaction signature**: a post-quantum algorithm can quickly generate a user’s private key starting from the signature of a transaction.

• **Block hashes**: a post-quantum algorithm can be used by an attacker to break the hash linking between blocks, writing new data in a block and quickly rebuilding the entire chain.
Changes in GO code

• Geth implementation – v1.11.0.

• Support multiple signature algorithms.

• crypto.go and signature_cgo.go files.
Changes in GO code

• We create `crypto_ecdsa.go` and `signature_cgo_ecdsa.go`.

• Constraint: ECDSA-compliant algorithm required.

• Example: generation of a key-pair using the interface.

```go
/* Generation of a key-pair using an interface */
func GenerateKey(*privateKey, error) { 
    switch actualAlgorithm { 
    case "ECDSA":
        return crypto_ecdsa.GenerateKey()
    case "SPHINS":
        return crypto_sphincs.GenerateKey()
    case "OTHER":
        return crypto_other.GenerateKey()
    default:
        return error("Unknown algorithm")
    }
}
```
Validation

• A node has been run and tested using the interface.

• Connection to Ethereum mainnet and transaction submission in Sepolia testnet.
Conclusion

• Interface implementation of an Ethereum node for supporting different signature methods.

• Post-quantum resistant Blockchain.

• Future work: extension of the interface to support algorithms with a different structure than ECDSA.
Thank you for your attention

marco.fiore@poliba.it  @Mackerkun