Secure distributed database with changes logging

Anton Bozhedarov
Vsevolod Glazov
Ivan Matvienko
Nikolay Shvetsov
Andrey Vlasov
Anton Razzhigaev

Problem Actuality (Relevance)

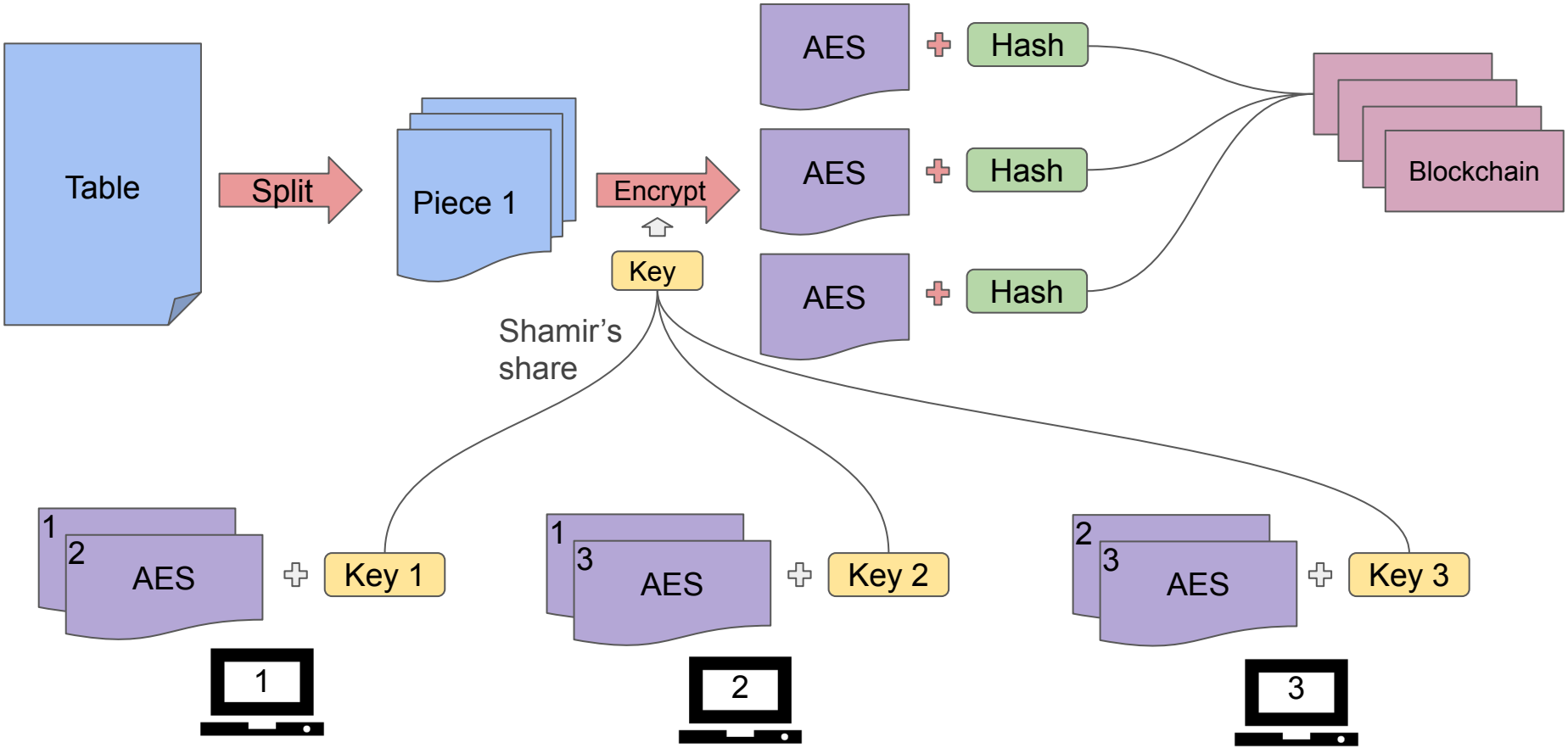
Our solution combines two technologies:

- ➔ Distributed database increases security against unauthorized access and robustness in case of failure of a part of the system.
- ➔ Blockchain technology allows to keep data secured in a system in which there is no trust to all participants. This is achieved by coordinating any change in the database by most participants.

Architecture

- ➔ File Encryption and Distribution
- ➔ Document Editing Request
- ➔ Change Confirmation
- ➔ Block Validation

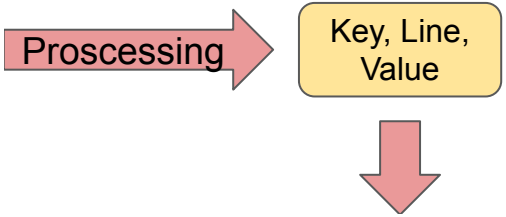
File Distribution

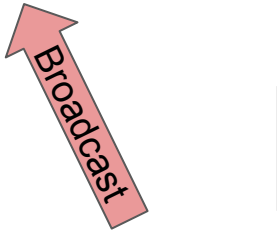


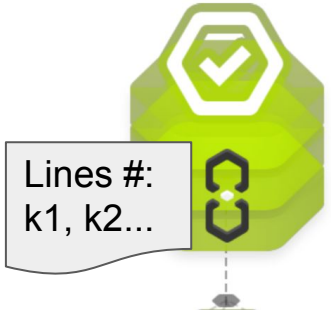
Blockchain schema

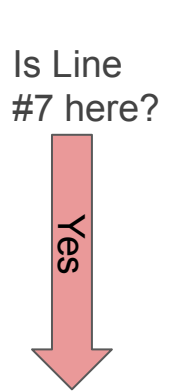
Request transaction

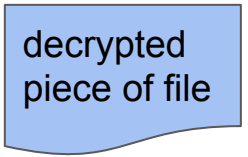
Input	Line #: 7 New Value: “I love EXONUM” Keys: [“...”,”...”,”...”]
Output	File decryption key: [“...”] New Value: “I love EXONUM” Line #: 7

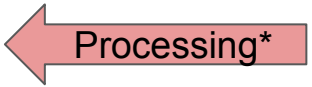












Validating transactions

Input	Request tx New Value: “I love EXONUM” Hash of decrypted piece of file Hash of encrypted piece of file
Output	True

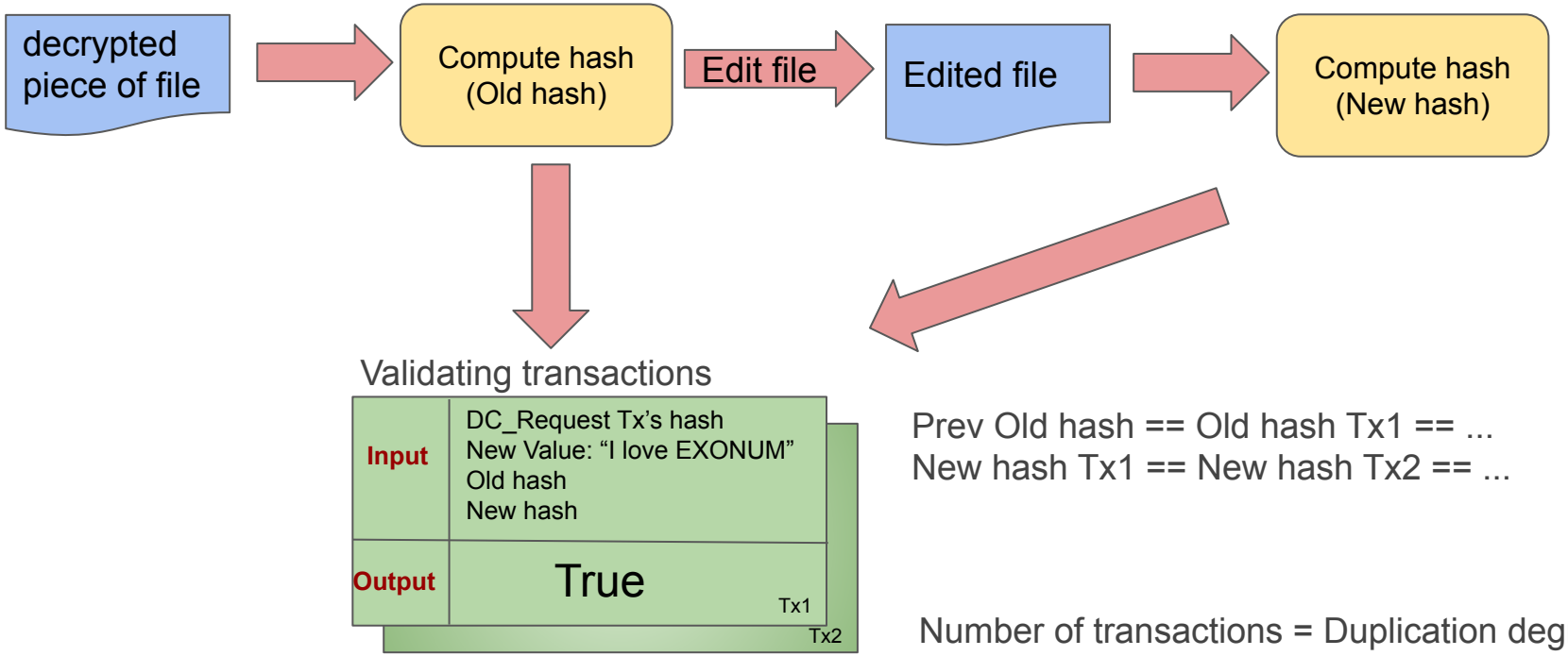
Tx1

Tx2

Block

- Request transaction
- Validating transactions
- prev_hash
- state_hash

Creation of Validating Tx



Implementation issues

- **Exonum**
  - Rust (using Cargo)
  - Complicated Architecture
  - Documentation is not clear and comprehensive at all
- **API**
  - The only way to get some information is to implement API
  - Blockchain is isolated from outer world
- **Protocol Buffers**
  - Some required types weren’t implemented
- **Front-end client**
  - JavaScript

Exonum

```rust
#[derive(Serialize, Deserialize, Clone, Debug, ProtobufConvert)]
#[exonum(pb = "proto::User")]
pub struct User {
    /// Public key of user.
    pub pub_key: PublicKey,
    /// Name of the User.
    pub name: String,
    /// Share of the common secret according to Shamir Sharing Scheme.
    pub key_shard: u64,
}

/// Additional methods for managing users in an immutable fashion.
impl User {
    /// Create new User.
    pub fn new(&pub_key: &PublicKey, name: &str, key_shard: u64) -> Self {
        Self {
            pub_key,
            name: name.to_owned(),
            key_shard,
        }
    }
```

Exonum

```rust
/// Transaction group.
#[derive(Serialize, Deserialize, Clone, Debug, TransactionSet)]
pub enum StorageTransactions {
    /// Create users transaction.
    CreateUser(TxCreateUser),
    /// Request of Document change transaction.
    ChangeDocument(TxChangeDocument),
    /// Validate Changes transaction
    ValidateChanges(TxValidateCahnges)
}
```

Exonum

```
/// Implementation of Shamir Common Secret recovery
fn Shamir (key_shards: [u64], values: [u64]) -> u64 {…
}

impl Transaction for TxChangeDocument {
    /// Retrieves document lines and line numbers to change; After that we apply
    /// Shamir method (function) ti calculate the common secret key, that TxChangeDocument should
    /// return. Also new line, and line numbers to change should be an outbut of that Tx to be processed
    /// in further steps of our scheme in python script

    fn execute(&self, mut context: TransactionContext) -> ExecutionResult {
        let comon_secret = Shamir(key_shards, values);
        lines_number;
        new_line;
     }
    /// Verification of transaction
    fn verify(&self, mut context: TransactionContext) -> ExecutionResult {…
    }
}
```

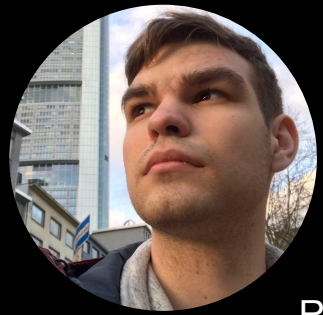
Results

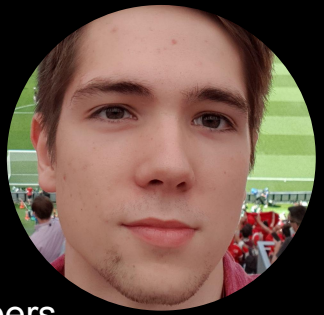
- **AES** encryption implemented
- **Shamir scheme** for key sharing used
- **Exonum** blockchain network implemented
- An **architecture** for secure database designed
- Had unbelievable amount of **fun**

Architecture

Cryptography

No Questions!





Rust Developers