**The TrueChain Smart Contracts - Best Practices**

This document is intended to provide some development experience for TrueChain smart contract developers. Including smart contract coding specifications, security development concepts, etc

**Coding Standards**

Every language has its own coding specification. Solidity is the smart contract development language currently supported by the truechain. Let us take this as an example.

**Naming conventions**

Contracts, libraries, events, enumerations, and structure naming should be capitalized using initials . For example: MyContract, TransferToken

Functions, parameters, variables, and decorator naming should use the lowercase of first letter . For example: getBalance, initialSupply, onlyOwner

**Code format**

Indent using four spaces instead of Tab.

There should be blank lines between functions or contracts.

When defining contracts, functions, libraries, structures, it is recommended that the left parenthesis be defined on one line.

A single-line expression to avoid unnecessary spaces

Each line should not be too long, controlled within 79 characters

**Function**

Recommended order of functions:

Constructor Function

Fallback Function (If needed)

External function

Public function

Internal Function

Private Function

Explicit the visibility of the function.

Function visibility is in front of the modifier. For example: public onlyOwner

The definition of functions and events should be clearly distinguishable. For example: the function is transferToken and the event is LogTransfer.

**Safety Guidelines**

Developing a smart contract requires a new engineering mindset that is different from the development of our previous projects. Once a smart contract is deployed on a blockchain network, it will be immutable and will be permanent. So the safe development of smart contracts requires more care. Here are some ideas for security development. I hope to help you avoid writing some unsafe development code.

**Facing error coding**

A good practice is to structure functions as: condition, behavior, interaction

First, check all pre-conditions

Then, modify the status of the contract

Finally, interact with other contracts.

**Discreetly released**

Try to find and fix possible bugs before officially releasing smart contracts

Thoroughly test smart contracts and test them in time after any new attack methods are discovered (including already released contracts)

The bug bonus plan is provided from the release of the original version on the testnet.

Phased release, providing adequate testing at each stage

**Keep it simple**

Complex smart contracts increase the risk of errors.

Ensure smart contract logic is simple

Ensure contract and function modularity

Use contracts or tools that are already widely used (for example, math libraries)

If conditions permit, it is clearer that performance is more important than performance.

Use blockchain only in the decentralized part of the system

**Keep Updating**

Make sure you get the latest security progress.

Check your smart contract when any new vulnerabilities are discovered

Update the used libraries or tools to the latest as soon as possible

Use the latest security technology

**Understand the characteristics of blockchain**

Be especially careful about calls to external contracts, because it is possible to execute a piece of malicious code and then change the control flow

It is clear that the public function is public, meaning it can be called maliciously. At the same time, private data is also visible to others.

Know the cost of gas and the gas limit of the block