Desarrollo Blockchain Ethereum con Solidity

Módulo 3 - ABI y Metadatos

ABI y Metadatos

ABI

- La **Application Binary Interface** *(Interfaz Binaria de Aplicación)* o ABI es el modo estándar de interactuar con contratos en el ecosistema Ethereum, tanto desde fuera de la blockchain como en interacciones contrato-contrato.
- Los datos se codifican siguiendo su tipo acorde a esta especificación.
- La ABI está **fuertemente tipada**, es conocida en tiempo de compilación y es estática.
- Los contratos tendrán las definiciones de la interfaz de cada contrato que vayan a llamar en tiempo de compilación.
- Esta especificación no abarca los contratos cuya interfaz sea dinámica o conocida exclusivamente en tiempo de ejecución. Estos casos, de volverse importantes, podrían manejarse adecuadamente como servicios construidos dentro del ecosistema Ethereum.

Contract Metadata

- El compilador Solidity genera automáticamente un archivo JSON que contiene información del contrato actual.
- Se puede usar este archivo para consultar la **versión** del compilador.
- Permite ver las fuentes utilizadas.
- Expone la documentación de **ABI** para interactuar de forma más segura con el contrato.

- Permite verificar su código fuente.
- Se genera a través del comando **solc –metadata**
- Se almacena en un archivo llamado **NombreDelContrato_meta.json**

Contract Metadata

A continuación, un ejemplo de Metadata para un smart contract.

```
{
  // Required: The version of the metadata format
  version: "1",
  // Required: Source code language, basically selects a "sub-version"
  // of the specification
  language: "Solidity",
  // Required: Details about the compiler, contents are specific
  // to the language.
  compiler: {
    // Required for Solidity: Version of the compiler
    version: "0.4.6+commit.2dabbdf0.Emscripten.clang",
    // Optional: Hash of the compiler binary which produced this output
    keccak256: "0x123..."
  },
  // Required: Compilation source files/source units, keys are file names
  sources:
  {
    "myFile.sol": {
      // Required: keccak256 hash of the source file
      "keccak256": "0x123...",
      // Required (unless "content" is used, see below): Sorted URL(s)
      // to the source file, protocol is more or less arbitrary, but a
      // Swarm URL is recommended
      "urls": [ "bzzr://56ab..." ]
    },
    "mortal": {
      // Required: keccak256 hash of the source file
      "keccak256": "0x234...",
      // Required (unless "url" is used): literal contents of the source file
      "content": "contract mortal is owned { function kill() { if (msg.sender == owner) selfdes
    }
  },
```

Solidity Assembly

- Solidity define un lenguaje de assembly que permite su utilización sin Solidity, llamado **inline-assembly.**
- Es posible intercalar declaraciones de Solidity con inline-assembly en un lenguaje más cercano al que soporta la EVM.
- Es posible ver la especificación completa junto a todos los opcodes asociados al momento en: https://solidity.readthedocs.io/en/latest/assembly.html

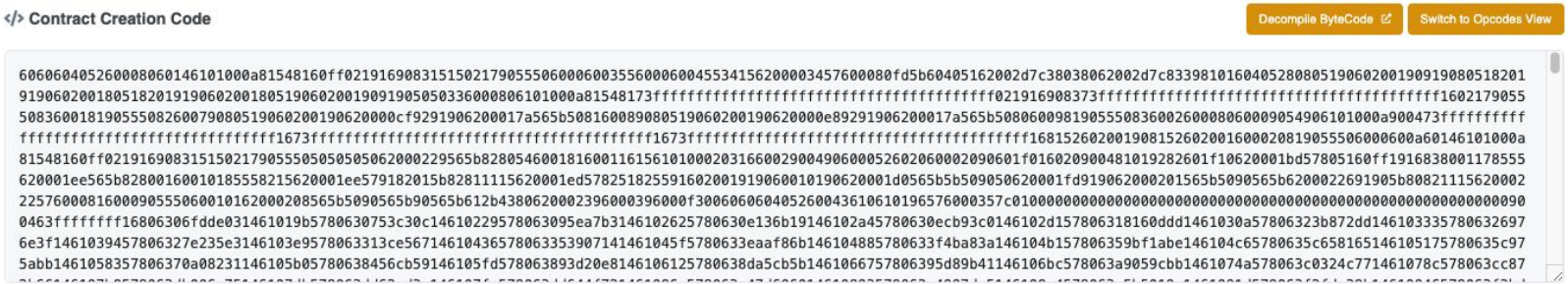
Solidity Assembly

- functional-style opcodes: `mul(1, add(2, 3))`
- assembly-local variables: `let x := add(2, 3)  let y := mload(0x40)  x := add(x, y)`
- access to external variables: `function f(uint x) public { assembly { x := sub(x, 1) } }`
- loops: `for { let i := 0 } lt(i, x) { i := add(i, 1) } { y := mul(2, y) }`
- if statements: `if slt(x, 0) { x := sub(0, x) }`
- switch statements: `switch x case 0 { y := mul(x, 2) } default { y := 0 }`
- function calls:

```
function f(x) -> y { switch x case 0 { y := 1 } default { y := mul(x, f(sub(x, 1))) }   }
```

Ingeniería inversa

- Dado a que lo que se almacena en el nodo es un bytecode que representa las operaciones del código, es posible revertir el mismo a código escrito en Solidity mediante algunos mecanismos.
- Por este motivo no se recomienda almacenar información crítica en el código fuente como contraseñas o credenciales.
- Ehterscan, uno de los exploradores más populares, permite decompilar un bytecode.



¡Muchas gracias!

¡Sigamos trabajando!