ABI 인코딩 엿보기

ABI (Application Binary Interface)

SmartContract 의 함수와 파라미터에 대한 MetaData를 정의해
Contract의 객체를 만들 수 있고 Contract의 함수를 호출할 수 있는 표준방법입니다.

Contract의 함수를 호출하기 위해서는 ABI Spec 에 맞게 데이터를 변환해야 합니다.
데이터는 크게 함수에 대한 정보와 함수 호출에 사용할 인수로 구분합니다.

- 함수에 대한 정보 : 함수의 이름과 인수에 대한 type을 정의 (signature)
- 인수 : 함수에서 사용할 인수

함수에 대한 정보

데이터의 처음 4바이트는 호출할 함수를 지정합니다.
호출할 함수 이름과 인수에 대한 타입 목록을 Keccak (SHA-3)로 해쉬로 변환하고

```
// example
transfer(address,uint256) >> 0xa9059cbb2ab09eb219583f4a59a5d0623ade346d962bcd4e46b1
```

변환한 해쉬의 4 byte `a9059cbb` 를 사용합니다.
(인수에 대한 type은 공백을 사용하지 않고 단일 쉼표로 구분합니다.)

인수 인코딩

함수 signature 이 후, 5 byte 부터는 인코딩된 인수가 옵니다.

인수 type

인코딩할 인수는 다음 기본 type 들이 있습니다.

- uint uint32,uint8,uint256
- int: int32,int8,int256
- address: uint160

- bool: uint8 (0,1)
- bytes: bytes1 … bytes32
- string
- …

인코딩 구분

인수 인코딩은 크게 동적인 유형과 정적인 유형에 따라 달라집니다.

동적 type

bytes
string
T[]
T[k] any T및 any k > 0
(T1,…,Tk) any Ti is dynamic <= i <= k

정적 type

동적 유형 이외는 정적

인코딩 Spec

인수 인코딩에 대한 기본 정의는 다음과 같습니다.
head와 tail로 나누어져 있고 head 부분에는 인수(동적 유형)에 대한 할당 위치가 오고
tail은 인수를 인코딩한 값이 옵니다.

일반적인 정의

abi spec 문서에는 다음과 같이 정의되어 있습니다.

```
(T1,...,Tk) for k >= 0 and any types T1, …, Tk
enc(X) = head(X(1)) ... head(X(k-1)) tail(X(0)) ... tail(X(k-1))
```

그리고 인코딩 유형에 따라 head와 tail 값이 달라집니다.

```
where X(i) is the ith component of the value, and head and tail are defined for Ti
// 정적 type 인코딩
head(X(i)) = enc(X(i)) and tail(X(i)) = ""
// 동적 type 인코딩
```

```
head(X(i)) = enc(len(head(X(0)) ... head(X(k-1)) tail(X(0)) ... tail(X(i-1))))
tail(X(i)) = enc(X(i))
```

정적 type 의 경우 먼저, head는 인수 값을 인코딩한 값이 들어가고 tail은 빈 값이 들어갑니다.
동적 type 의 경우 head 는 (head part의 길이 + tail을 인코딩한 값의 길이) 값이 들어갑니다.
head part의 길이는 일반적으로 type의 개수 * 32byte 값을 인코딩합니다.

위의 정의만 봐서는 이해하기가 힘들어서 조금 인수의 개수를 줄여서 살펴봤습니다.
`(string)` type이 있고 'abc'라는 값을 갖는 인수를 인코딩한다고 하면 다음과 같이 인코딩 됩니다.

인수가 1개이기 때문에 head,tail 각각 하나의 값이 들어갑니다.
head에 해당하는 값에는 type이 1개로 1 * 32 = 32 를 인코딩한 값이 들어가게 됩니다.
head 의미는 32 byte 후부터 인수의 실제 데이터가 위치한다고 보면 됩니다.
그리고 tail에는 'abc'를 인코딩한 데이터가 들어가게 됩니다.

```
enc(X) = head(X(0)) tail(X(0))
head(X(0)) = enc(len(head(X(0)))) // type 1개, 1 * 32 = 32 인코딩
tail(X(0)) = enc(X(0))  // 'abc' 인코딩
```

type을 하나 더 추가해서 인코딩한다고 가정해보겠습니다
`(string, string)` type이 있고 'abc','def' 라는 값을 갖는 인수를 인코딩하면 다음과 같이 인코딩 됩니다.

```
enc(X) = head(X(0)) head(X(1)) tail(X(0)) tail(X(1))
head(X(0)) = enc(len(head(X(0)) head(X(1))))  // type 2개, 2 * 32 = 64 인코딩
head(X(1)) = enc(len(head(X(0)) head(X(1)) tail(X(0)))) // head part length(64) + '
tail(X(0)) = enc(X(0))  // 'abc' 인코딩
tail(X(1)) = enc(X(1))  // 'def' 인코딩
```

example

실제 Contract 에 대한 함수 호출에 대한 인코딩하는 예를 살펴보도록 하겠습니다.

```
pragma solidity ^0.4.16;

contract Foo {
  function bar(bytes3[2]) public pure {}
  function baz(uint32 x, bool y) public pure returns (bool r) { r = x > 32 || y; }
```

```
    function sam(bytes, bool, uint[]) public pure {}
}
```

baz(uint32 x, bool y)

`baz`는 uint32와 bool로 정적인 type 을 인수로 갖는 함수입니다.
정적인 type이기 때문에 `head(X(i)) = enc(X(i)) and tail(X(i)) = ""` 로 인코딩이 하면 됩니다.
`69`, `true` 인수로 함수를 호출한다고 하면 인코딩 데이터는 다음과 같습니다.

```
// 인코딩 데이터
0xcdcd77c000000000000000000000000000000000000000000000000000000045
0000000000000000000000000000000000000000000000000000000000000001
```

0xcdcd77c0

첫 4 byte는 `baz(uint32,bool)` 함수의 signature 입니다.

0x0000000000000000000000000000000000000000000000000000000000000045

`69`를 hex로 변환한 값입니다. hex로 변환 후, 32byte로 데이터를 채웁니다.

0x0000000000000000000000000000000000000000000000000000000000000001

bool type은 uint8 로 변환(1:true)해서 인코딩을 합니다. hex로 변환 후, 32byte로 데이터를 채웁니다.(left padding)

bar(bytes3[2])

`bar`는 bytes3 유형에 2라는 size를 갖는 정적 type 을 인수로 갖는 함수입니다.
인코딩 방법은 인수를 byte데이터로 변환합니다.
`["abc", "def"]` 인수로 함수를 호출한다고 하면 인코딩 데이터는 다음과 같습니다.

```
// 인코딩 데이터
0xfce353f661626300000000000000000000000000000000000000000000000000
6465660000000000000000000000000000000000000000000000000000000000
```

0xfce353f6

첫 4 byte는 `bar(bytes3[2])` 함수의 signature 입니다.

0x6162630000000000000000000000000000000000000000000000000000000000

인수(array)의 첫번째 데이터 "abc"를 인코딩한 값입니다. string을 utf-8 byte 데이터로 변환하고 32byte로 데이터를 채웁니다. (왼쪽으로 정렬)

0x6465660000000000000000000000000000000000000000000000000000000000

인수(array)의 두번째 데이터 "def"를 인코딩한 값입니다. string을 utf-8 byte 데이터로 변환하고 32byte로 데이터를 채웁니다. (왼쪽으로 정렬)

sam(bytes, bool, uint[])

`sam` 에는 bytes와 uint[] 라는 동적 type과 bool이라는 정적 type을 인수로 갖고 있습니다.
정적 type과 다르게 동적 type 의 경우에는 head와 tail로 나누어 인코딩합니다.
head 부분에 동적타입 인수의 경우 실제 데이터 시작 위치를 인코딩하고 정적타입의 경우 데이터를 바로 인코딩합니다.
`"dave", true, [1,2,3]` 인수로 함수를 호출한다고 하면 다음과 같이 인코딩을 하면 됩니다.

```
enc(X) = head(X(0)) head(X(1)) head(X(2)) tail(X(0)) tail(X(1)) tail(X(2))
head("dave") = enc(len(head("dave") head(true) head([1,2,3])))
head(true) = enc(true)
head([1,2,3])) = enc(len(head("dave") head(true) head([1,2,3]) tail("dave") tail(tr
tail("dave") = enc("dave")
tail(true) = "" // 정적 타입이므로 empty string
tail([1,2,3]) = enc([1,2,3])
```

실제 인코딩한 데이터를 보면 다음과 같습니다.

```
0xa5643bf2000000000000000000000000000000000000000000000000000000000000006000000000000
```

0xa5643bf2

`sam(bytes,bool,uint[])` 함수의 signature 입니다.

0x0000000000000000000000000000000000000000000000000000000000000060

"dave" 인코딩 데이터의 위치를 인코딩한 값이 옵니다.
96byte 위치부터 "dave"의 인코딩 데이터가 위치합니다.

0x0000000000000000000000000000000000000000000000000000000000000001

bool type 은 정적 type 으로 true 를 인코딩한 값이 옵니다.

0x00000000000000000000000000000000000000000000000000000000000000a0

[1,2,3] 인수의 데이터 위치를 인코딩한 값이 옵니다.
160byte 위치부터 [1,2,3]의 인코딩 데이터가 위치합니다.

0x0000000000000000000000000000000000000000000000000000000000000004

"dave"의 인코딩 데이터가 시작합니다. string 인코딩은 bytes 변환 후,
(bytes length 인코딩 + bytes) 데이터가 옵니다. length는 32byte 로 인코딩합니다.

0x6461766500000000000000000000000000000000000000000000000000000000

인수 "dave"를 인코딩한 값입니다. string을 utf-8 byte 데이터로 변환하고
32byte로 데이터를 채웁니다. (왼쪽으로 정렬)

0x0000000000000000000000000000000000000000000000000000000000000003

[1,2,3] 인코딩 데이터가 시작합니다. array 인코딩은
(array에 대한 요소의 개수 인코딩 + 각 요소 인코딩) 데이터가 옵니다.
요소의 개수는 32byte 로 인코딩합니다.

0x0000000000000000000000000000000000000000000000000000000000000001, ...

array의 순서대로 데이터를 인코딩합니다. `uint` 타입의 인수를 인코딩한 값입니다.

참고

Ethereum-Contract-ABI