

한 번에 끝내는 블록체인 개발 A to Z

Chapter 2

Blockchain 2.0 - Ethereum

Chapter 2

Blockchain 2.0 - Ethereum

Transaction Structure

Transaction

구조

- Transaction은 EOA가 EOA에게 Eth를 전송하거나 EOA가 CA를 호출할 때 사용되는 구조이다. 이 데이터는 블록체인 상에 기록된다.

```
type TxData interface {  
    txType() byte // returns the type ID  
    copy() TxData // creates a deep copy and initializes all fields  
  
    chainID() *big.Int  
    accessList() AccessList  
    data() []byte  
    gas() uint64  
    gasPrice() *big.Int  
    gasTipCap() *big.Int  
    gasFeeCap() *big.Int  
    value() *big.Int  
    nonce() uint64  
    to() *common.Address  
  
    rawSignatureValues() (v, r, s *big.Int)  
    setSignatureValues(chainID, v, r, s *big.Int)  
}
```

(출처 : <https://github.com/ethereum/go-ethereum/blob/master/core/types/transaction.go>)

EOA-EOA Transaction

- EOA-EOA간의 거래는 Value에 보내는 Eth양이 들어가고 Input Data에는 빈 값이 들어간다.

Transaction Hash:	0x6287ce2de7e97ad3cd0043c57b661cd20b849e0df022cbe7606330983af5e9b5
Status:	Success
Block:	15106660 1 Block Confirmation
Timestamp:	12 secs ago (Jul-09-2022 06:09:06 AM +UTC) Confirmed within 30 secs
From:	0x600b2104a3be02fd8c9e2c6c3eddd72120b832e1
To:	0x50f180e794a3fe852c7b8eb88982a2eb45e6c2b5
Value:	1.53845466365969408 Ether (\$1,867.75)
Transaction Fee:	0.000232876569156 Ether (\$0.28)
Gas Price:	0.000000011089360436 Ether (11.089360436 Gwei)
Gas Limit & Usage by Txn:	21,000 21,000 (100%)
Gas Fees:	Base: 10.902292912 Gwei
Others:	Txn Type: 0 (Legacy) Nonce: 4 Position: 106
Input Data:	0x

(출처 : etherscan.io)

EOA-CA

Transaction

- EOA-CA 간의 거래는 Value에는 보내는 값이 없는 경우에 0이 들어가고 Data에 호출하는 함수명과 파라미터 값이 들어간다.

Transaction Hash:	0xcdcd6a3d1ecae72470352daf9cc3868d5acfc30b6be2a86008f4a8b73005fac2 🔗
Status:	Success
Block:	15106664 1 Block Confirmation
Timestamp:	50 secs ago (Jul-09-2022 06:09:47 AM +UTC) Confirmed within 30 secs
From:	0x5e8c6592205e938cedcad7c8d4b63589a1a54166 🔗
Interacted With (To):	Contract 0xdac17f958d2ee523a2206206994597c13d831ec7 (Tether: USDT Stablecoin) 👍 🔗
Tokens Transferred:	From 0x5e8c6592205e9... To 0x6b9353bf4f90ab... For 135.31 (\$135.45) 👍 Tether USD (USDT)
Value:	0 Ether (\$0.00)
Transaction Fee:	0.000566652100402311 Ether (\$0.69)
Gas Price:	0.000000012289403379 Ether (12.289403379 Gwei)
Gas Limit & Usage by Txn:	90,704 46,109 (50.83%)
Gas Fees:	Base: 11.289403378 Gwei Max: 18.676284944 Gwei Max Priority: 1.000000001 Gwei
Burnt & Txn Savings Fees:	🔥 Burnt: 0.000520543100356202 Ether (\$0.63) 👍 Txn Savings: 0.000294492722080585 Ether (\$0.36)
Others:	Txn Type: 2 (EIP-1559) Nonce: 3 Position: 273
Input Data:	<pre>Function: transfer(address _to, uint256 _value) MethodID: 0xa9059cbb [0]: 00 [1]: 00</pre>

(출처 : etherscan.io)

Transaction 서명

- Ethereum은 secp256k1기반의 타원곡선암호(ECC)를 이용하여 서명하고 검증한다.
- Transaction 에서 r, s 가 실제 서명이고 v 는 서명값 복구를 위한 값으로 27,28을 사용하고 있다.

서명절차

- 1) 서명할 데이터의 hash 값 e 를 구한다.
- 2) n 보다 작은 랜덤값 k 를 생성한다.
- 3) $k \cdot G$ 인 좌표 (x_1, y_1) 를 계산한다.
- 4) $r = x_1 \bmod n$ 인 서명 값 r 을 계산한다.
($r == 0$ 인 경우 2) 부터 다시 시작)
- 5) $s = k^{-1}(e + r d_a) \bmod n$ 인 두 번째 서명 값 s 를 계산한다.
($s == 0$ 인 경우 2) 부터 다시 시작)



검증절차

- 1) 원본 메시지의 hash 값 e 를 구한다.
- 2) EC 상에서 $R = (x_1, y_1)$ 인 좌표를 구한다.
($x_1 = r$ if $v=27$, $x_1 = r+n$ if $v=28$)
- 3) $u_1 = -zr^{-1} \bmod n$ 을 계산한다.
- 4) $u_2 = sr^{-1} \bmod n$ 을 계산한다.
- 5) 공개 키 $Q_a = (x_a, y_b) = u_1 \times G + u_2 \times R$ 을 구한다.
- 6) 공개 키에서 Address(주소)를 구한 뒤 송금자와 비교하여 일치하는지 확인한다.

Message

구조

- Message는 CA가 CA를 호출할때 발생하는 네트워크 구조이다. EVM 상에서만 존재하는 가상 객체이다. 블록체인상에 기록이 남지 않는다.

```
type Message struct {  
    to          *common.Address  
    from        common.Address  
    nonce       uint64  
    amount      *big.Int  
    gasLimit    uint64  
    gasPrice    *big.Int  
    gasFeeCap   *big.Int  
    gasTipCap   *big.Int  
    data        []byte  
    accessList  AccessList  
    isFake      bool  
}
```

(출처 : <https://github.com/ethereum/go-ethereum/blob/master/core/types/transaction.go>)

CA-CA Transaction

- CA-CA 간의 호출 정보는 Internal Transaction이라고 부르며 Contract 코드 상에서 Delegatecall, StaticCall, call 함수를 통해 발생한다.
- (CA-EOA ETH 전송, selfdestruct 포함)

Overview **Internal Txns** Logs (5) State Comments

Advanced The contract call From 0xa955875361912d9ac6... To 0x68b3465833fb72a70e... produced 11 Internal Transactions **Advanced**

Type	Trace Address	From	To	Value	Gas Limit
✓	delegatecall_0_1	0x68b3465833fb72a70e...	→ 0x68b3465833fb72a70e...	0 Ether	249,229
✓	call_0_1_1	0x68b3465833fb72a70e...	→ ↻ 0xc02aaa39b223fe8d0a...	0.02 Ether	234,035
✓	call_0_1_1	0x68b3465833fb72a70e...	→ ↻ 0xc02aaa39b223fe8d0a...	0 Ether	210,011
✓	staticcall_0_1_1	0x68b3465833fb72a70e...	→ ↻ 0x2139f1728278aad927...	0 Ether	198,891
✓	staticcall_0_1_1	0x68b3465833fb72a70e...	→ ↻ 0x17a2642021afceefd43...	0 Ether	191,916
✓	staticcall_0_1_1	0x68b3465833fb72a70e...	→ ↻ 0xc02aaa39b223fe8d0a...	0 Ether	188,599
✓	call_0_1_1	0x68b3465833fb72a70e...	→ ↻ 0x17a2642021afceefd43...	0 Ether	186,398
✓	call_0_1_1_1	0x17a2642021afceefd43...	→ ↻ 0x2139f1728278aad927...	0 Ether	172,781
✓	staticcall_0_1_1_1	0x17a2642021afceefd43...	→ ↻ 0x2139f1728278aad927...	0 Ether	100,995
✓	staticcall_0_1_1_1	0x17a2642021afceefd43...	→ ↻ 0xc02aaa39b223fe8d0a...	0 Ether	99,674
✓	staticcall_0_1_1	0x68b3465833fb72a70e...	→ ↻ 0x2139f1728278aad927...	0 Ether	48,603

(출처 : etherscan.io)

Receipt 구조

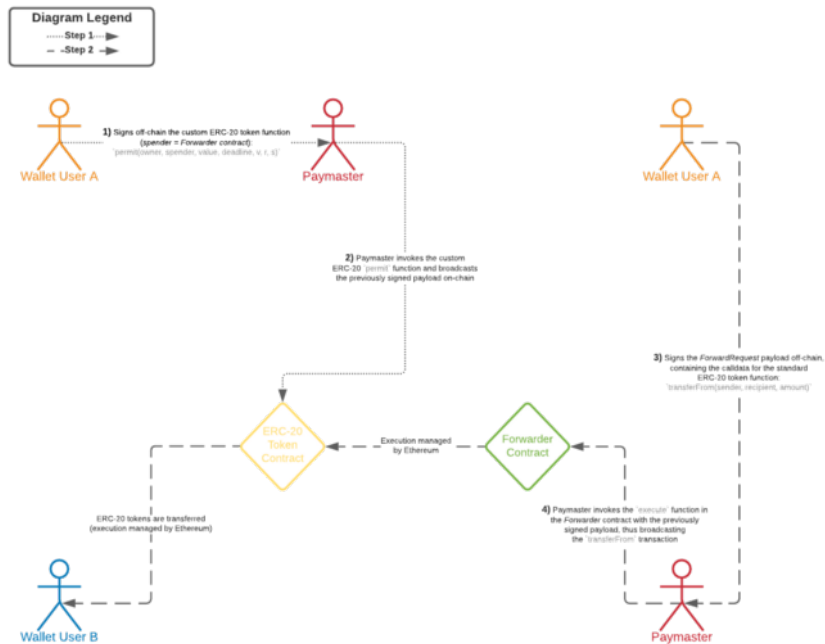
- EVM에서 Transaction을 실행하고 결과값이 저장되는 곳이다. 실제로 사용된 Gas와 컨트랙트 호출시에 발생하는 Log등이 저장되는 구조이다. 이 데이터는 블록체인 상 기록된다.

```
type Receipt struct {  
    // Consensus fields: These fields are defined by the Yellow Paper  
    Type      uint8   `json:"type,omitempty"`  
    PostState  []byte  `json:"root"`  
    Status     uint64   `json:"status"`  
    CumulativeGasUsed uint64 `json:"cumulativeGasUsed" gencodec:"required"`  
    Bloom      Bloom   `json:"logsBloom"          gencodec:"required"`  
    Logs       []*Log  `json:"logs"               gencodec:"required"`  
  
    // Implementation fields: These fields are added by geth when processing a transaction.  
    // They are stored in the chain database.  
    TxHash      common.Hash `json:"transactionHash" gencodec:"required"`  
    ContractAddress common.Address `json:"contractAddress"`  
    GasUsed     uint64      `json:"gasUsed" gencodec:"required"`  
  
    // Inclusion information: These fields provide information about the inclusion of the  
    // transaction corresponding to this receipt.  
    BlockHash      common.Hash `json:"blockHash,omitempty"`  
    BlockNumber    *big.Int   `json:"blockNumber,omitempty"`  
    TransactionIndex uint       `json:"transactionIndex"`  
}
```

(출처 : <https://github.com/ethereum/go-ethereum/blob/master/core/types/receipts.go>)

Meta Transaction

- Meta Transaction이란 특정 User의 거래를 대신 실행해주는 거래 방식이다. ERC20 토큰 전송을 할 때, 사용자가 ERC 20 CA 호출을 위해서 지불해야하는 ETH 수수료를 대행업체가 대신 납부할 수 있다.



(출처 : <https://github.com/pcaversaccio/metatx>)

EIP-2770

Code

- Smart Contract 코드 상에서 사용자의 서명 검증하는 부분이다. Transaction 서명 부분에서 진행했던 검증 방안과 동일한 방법으로 EVM 상에서 진행된다.

```
function verify(ForwardRequest calldata req, bytes calldata signature) public view returns (bool) {  
    address signer = _hashTypedDataV4(keccak256(abi.encode(  
        _TYPEHASH,  
        req.from,  
        req.to,  
        req.value,  
        req.gas,  
        req.nonce,  
        keccak256(req.data)  
    ))).recover(signature);  
    return _nonces[req.from] == req.nonce && signer == req.from;  
}
```

(출처 : <https://github.com/pcaversaccio/metatx/blob/main/contracts/Forwarder.sol>)

EIP-2770

Code

- 호출한 사용자의 거래가 검증이 완료되면
CA-CA간 거래인 call을 통해서 대행
Contract가 토큰 전송을 대신 호출하게
된다.

```
function execute(ForwardRequest calldata req, bytes calldata signature) public payable whenNotPaused() returns (bool, bytes memory) {
    require(_senderWhitelist[msg.sender], "AwlForwarder: sender of meta-transaction is not whitelisted");
    require(verify(req, signature), "AwlForwarder: signature does not match request");
    _nonces[req.from] = req.nonce + 1;

    // solhint-disable-next-line avoid-low-level-calls
    (bool success, bytes memory returndata) = req.to.call{gas: req.gas, value: req.value}(abi.encodePacked(req.data, req.from));

    if (!success) {
        // solhint-disable-next-line no-inline-assembly
        assembly {
            returndatacopy(0, 0, returndatasize())
            revert(0, returndatasize())
        }
    }

    /**
     * @dev Validates that the relayer/forwarder EOA has sent enough gas for the call.
     * See https://ronan.eth.link/blog/ethereum-gas-dangers/.
     */
    assert(gasleft() > req.gas / 63);

    emit MetaTransactionExecuted(req.from, req.to, req.data);

    return (success, returndata);
}
```

(출처 : <https://github.com/pcaversaccio/metatx/blob/main/contracts/Forwarder.sol>)