한 번에 끝내는
블록체인 개발 A to Z

Chapter 2

Blockchain 2.0 - Ethereum

Chapter 2

Blockchain 2.0 - Ethereum

Transaction Structure

Transaction 구조

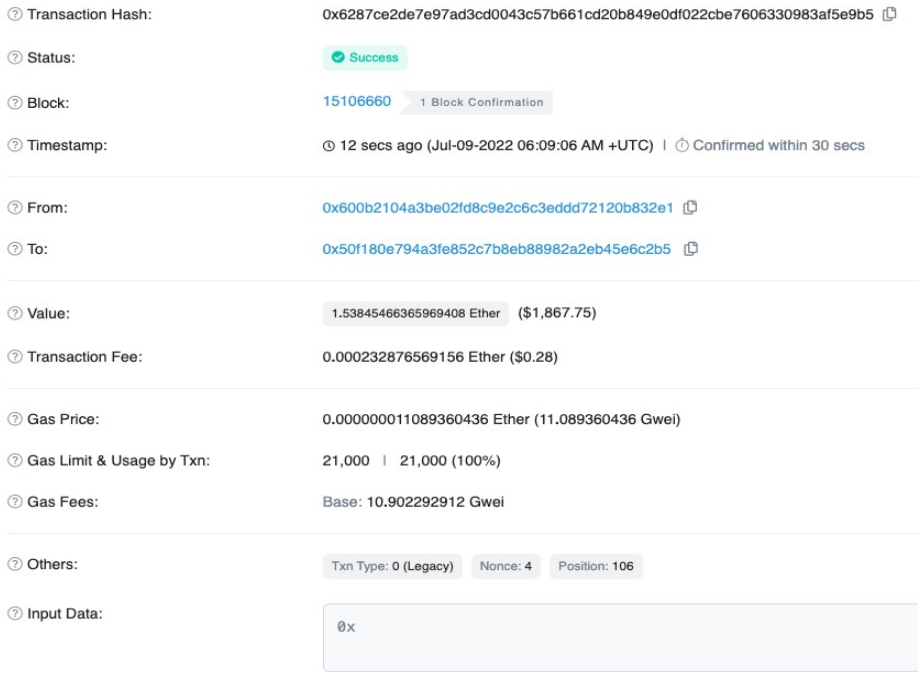
- Tranasction은 EOA가 EOA에게 Eth를 전송하거나 EOA가 CA를 호출할 때 사용되는 구조이다. 이 데이터는 블록체인 상에 기록된다.

```
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

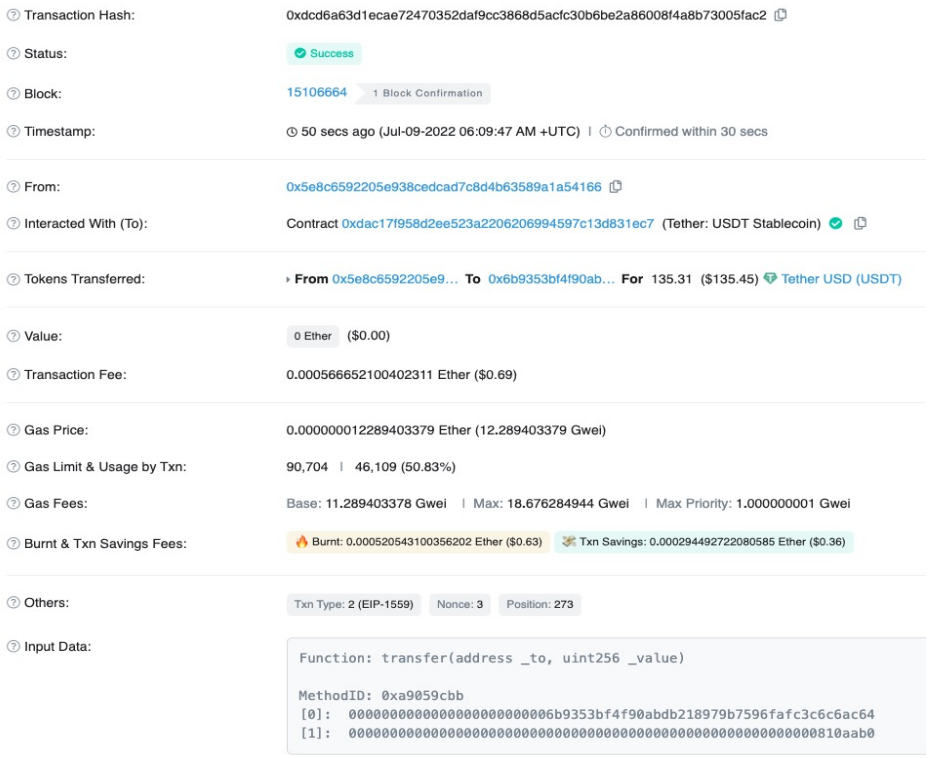
- EOA-EOA간의 거래는 Value에 보내는 Eth양이 들어가고 Input Data에는 빈 값이 들어간다.



(출처 : etherscan.io)

EOA-CA Transaction

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



(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 e를 구한다.
2) n 보다 작은 랜덤값 k를 생성한다.
3) k*G 인 좌표 (x1,y1)를 계산한다.
4) r = x1 mod n인 서명 값 r을 계산한다.
   (r == 0 인 경우 2) 부터 다시 시작)
5) $s = k^{-1}(e + rd_a) \bmod n$ 인 두 번째 서명 값 s를 계산한다.
   (s ==0 0 인 경우 2) 부터 다시 시작)

검증절차

1) 원본 메시지의 hash 값 e를 구한다.
2) EC 상에서 R =(x1, y1) 인 좌표를 구한다.
   (x1= r if v=27, x1= 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

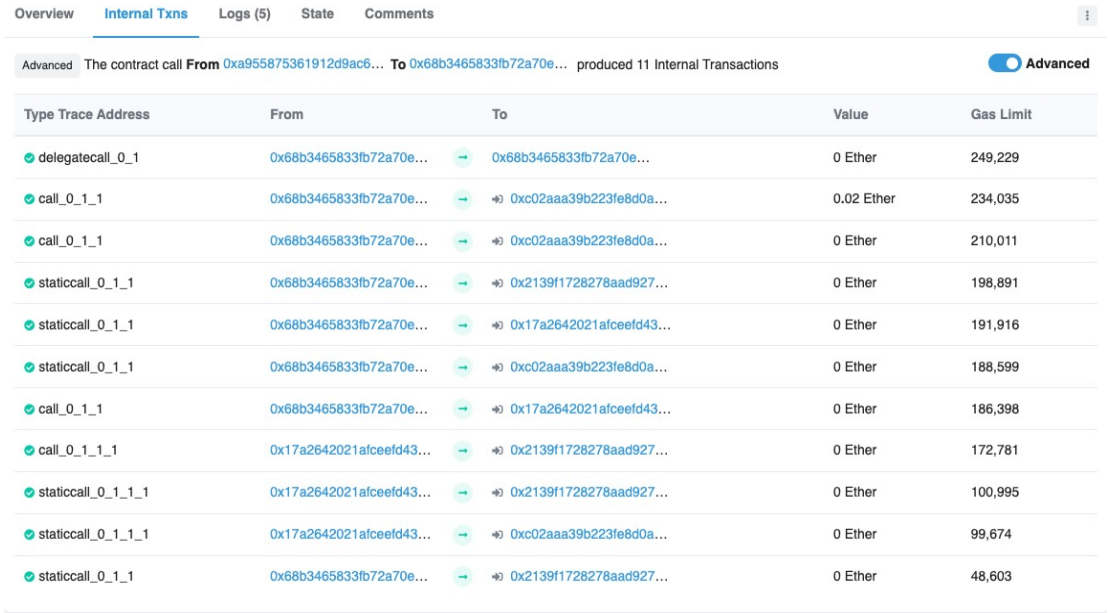
- Message는 CA가 CA를 호출할때 발생하는 네트워크 구조이다. EVM 상에서만 존재하는 가상 객체이다. 블록체인상에 기록이 남지 않는다.

```
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)



(출처 : etherscan.io)

Receipt 구조

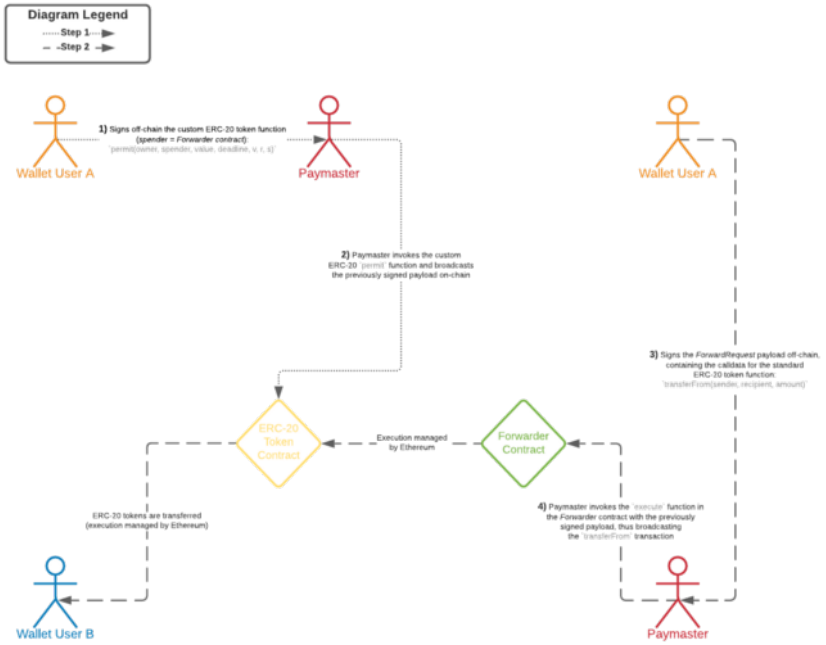
- EVM에서 Transaction을 실행하고 결과값이 저장되는 곳이다. 실제로 사용된 Gas와 컨트랙트 호출시에 발생하는 Log등이 저장되는 구조이다. 이 데이터는 블록체인 상 기록된다.

```
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)