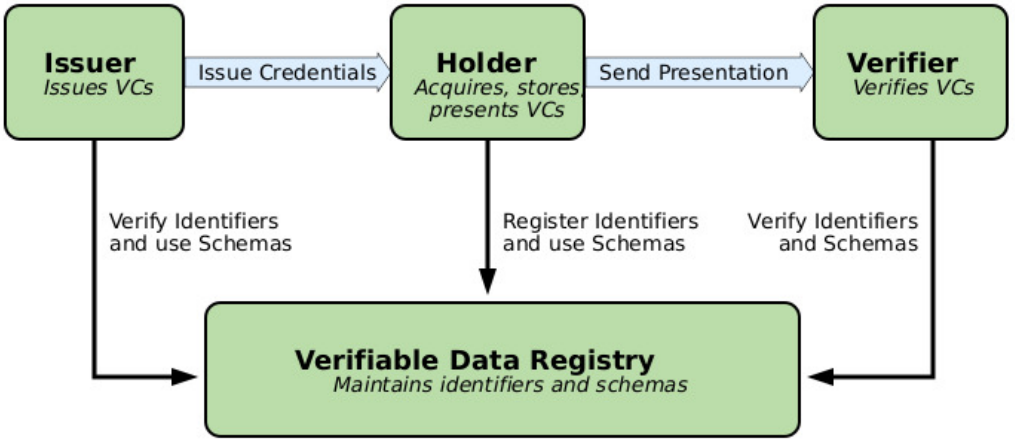
# DID 생성 실습

## 개요




1. Holder

   개인정보 데이터를 보유한 개인. Holder만이 자신의 데이터를 소유할 수 있으며, 관련 기관에 자신과 관련된 데이터의 export를 요청할 수 있다.

2. Issuer

   Holder와 관련된 데이터들을 발급해주는 기관이다. 예를 들어, holder의 졸업증명서를 발급해주는 대학교가 될 수 있고, 주민 드록 등본을 발급해주는 행정기관이 될 수도 있다,

3. Verifier

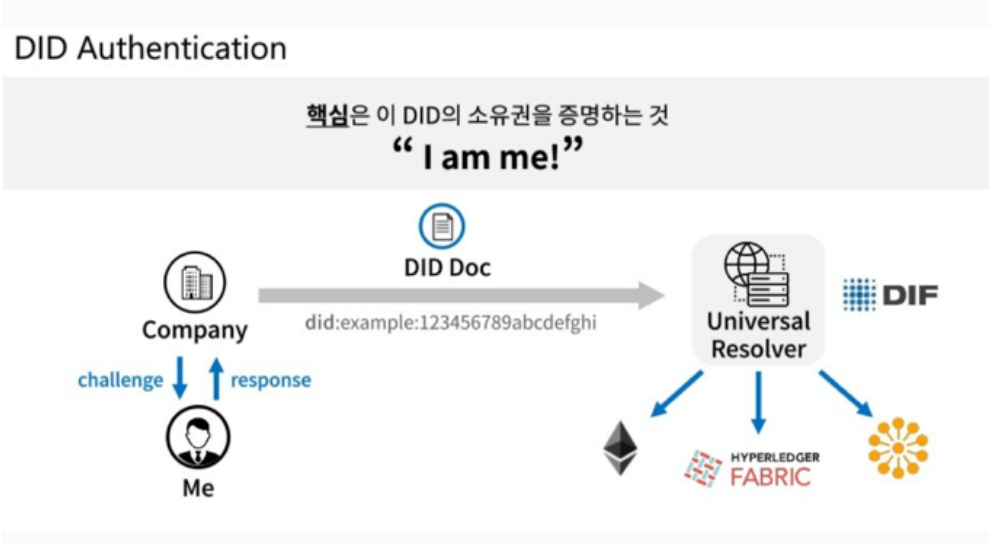
   Holder의 개인정보 데이터가 필요한 기관이며, holder가 제출한 정보를 검증하는 기관이이다. ex) holder가 취업하기를 원하는 회사.

4. Data registry

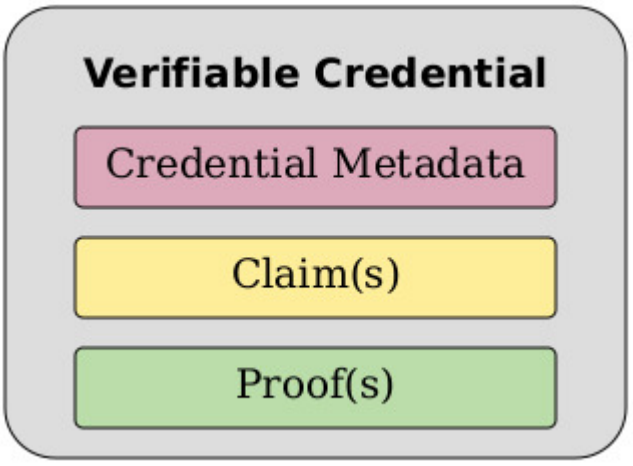
   DID에 대한 정보를 저장/관리하는 곳, DID 시스템의 참여자는 이곳의 정보를 통해서 검증이 가능하다.

## DID 형태


```
{
  "@context": [
    "https://www.w3.org/ns/did/v1",
    "https://w3id.org/security/suites/ed25519-2020/v1"
  ]
  "id": "did:example:123456789abcdefghi",
  "authentication": [{
    "id": "did:example:123456789abcdefghi#keys-1",
    "type": "Ed25519VerificationKey2020",
    "controller": "did:example:123456789abcdefghi",
    "publicKeyMultibase": "zH3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXmqPV"
  }]
}
```




## VC & VP



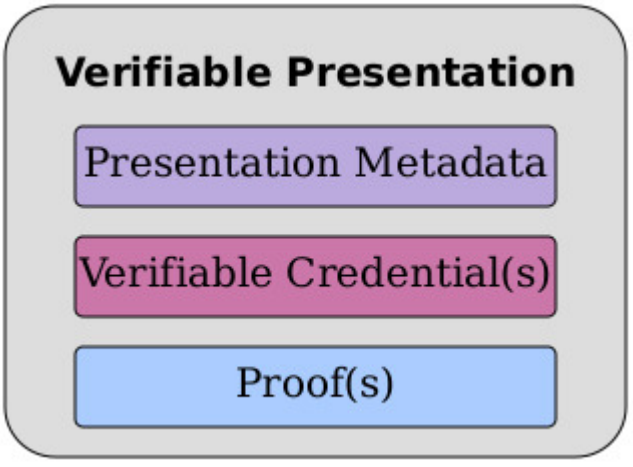

1. Verifiable Credential

   holder가 요청한 데이터 항목들에 대해, issuer가 데이터를 정제하여 holder에게 전달하는 데이터 포맷이다.

   각각의 데이터는 Claim으로 존재하는데, 단일 claim 또는 다수의 claim들을 VC에서 보유할 수 있다.

   Holder가 졸업증명서를 요청하였다면 issuer는 졸업증명서에 대한 정보를 claim으로 만들어 VC에 포함시킨다.

   VC에는 proof가 존재하며, 이는 issuer가 발급한것이 맞다는 것을 보장하기 위한 issuer의 DID private key로 생성된 서명이다.



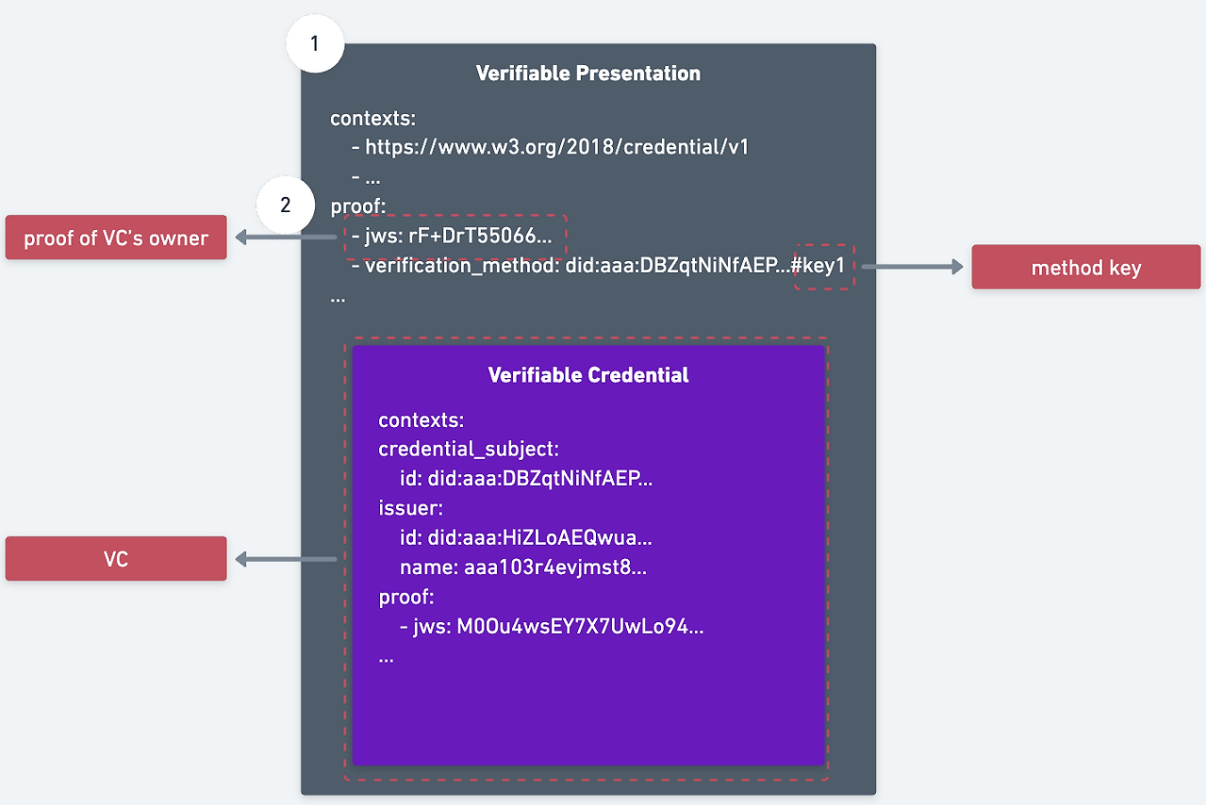

2. Verifiable Presentation

   Holder는 자신이 보유한 VC를 verifier에게 제출할 때 VP로 만들어야한다.

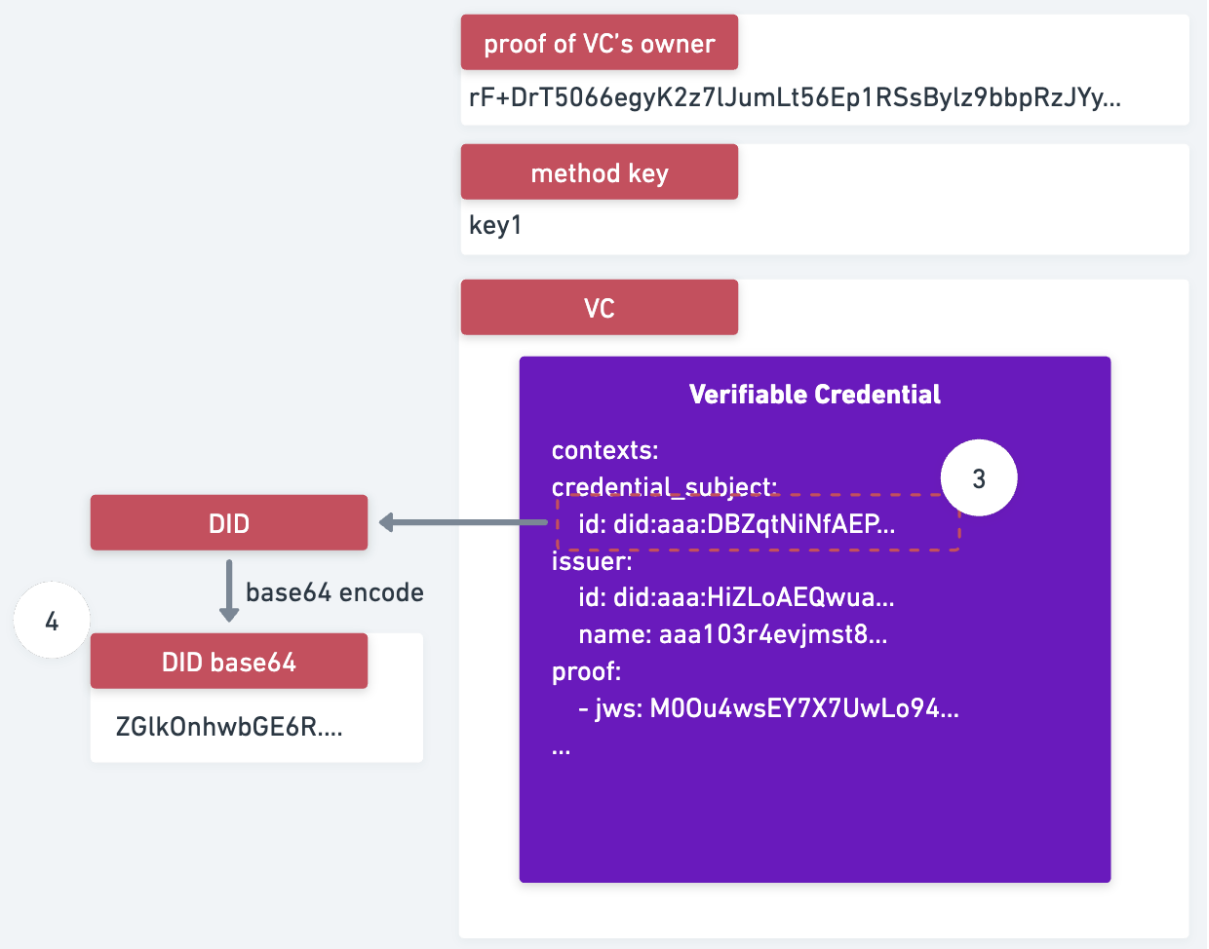
   SSI의 특성에 맞게 보유한 모든 데이터를 전송하는 것이 아닌, verifier가 필요한 데이터를 선택하여 단일 혹은 다수의 VC를 모아서 VP로 만든다.

VP에는 VC와 비슷하게 Proof가 존재하는데, 이는 VP를 생성하는 것이 Holder가 맞다는 것을 보장하기 위한 holder의 DID private key로 생성된 서명이다.

## DID Process

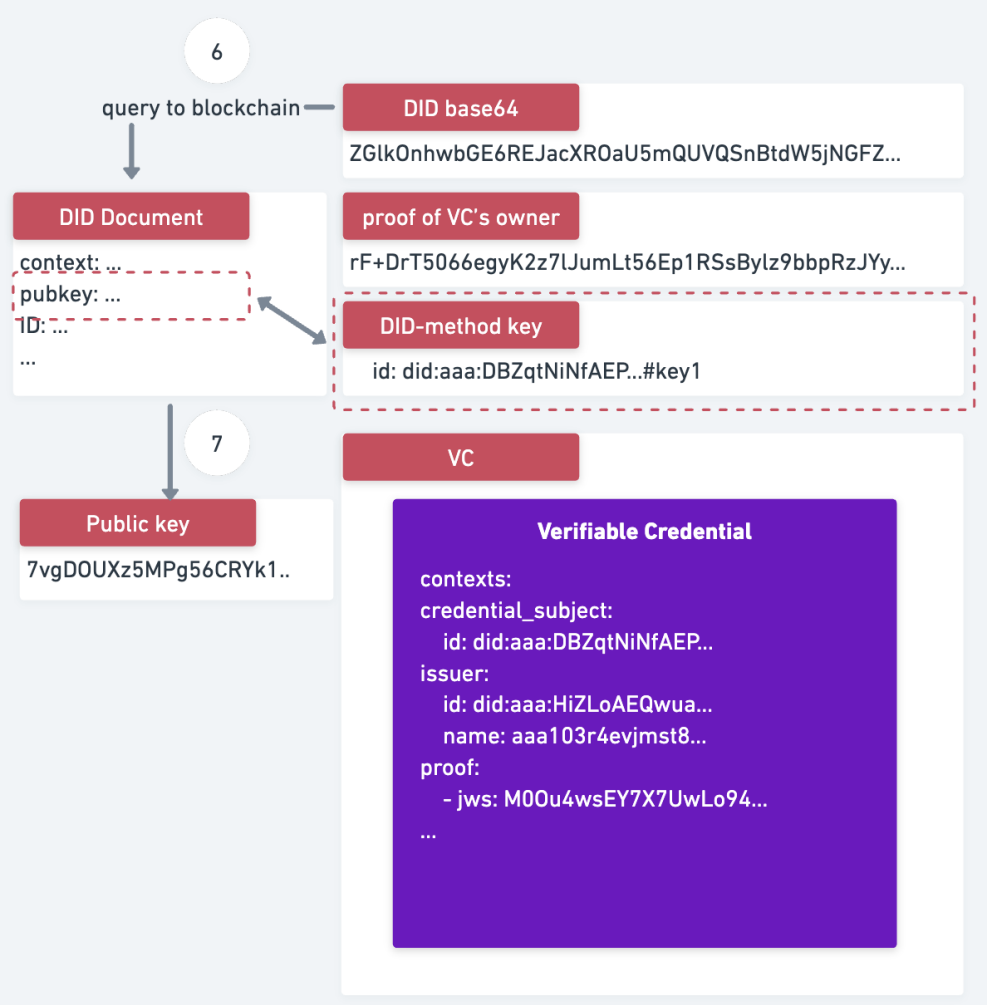



1. Holder로 부터 VP를 수신
2. Verifier는 제출된 VP를 파싱한다. Holder의 서명을 검증하기 위해 확인해야할 사항은 VP 내의 VC, VC(VP)의 소유자의 proof, VP proof를 만들기 위해 사용된 사용자의 DID method key이다.

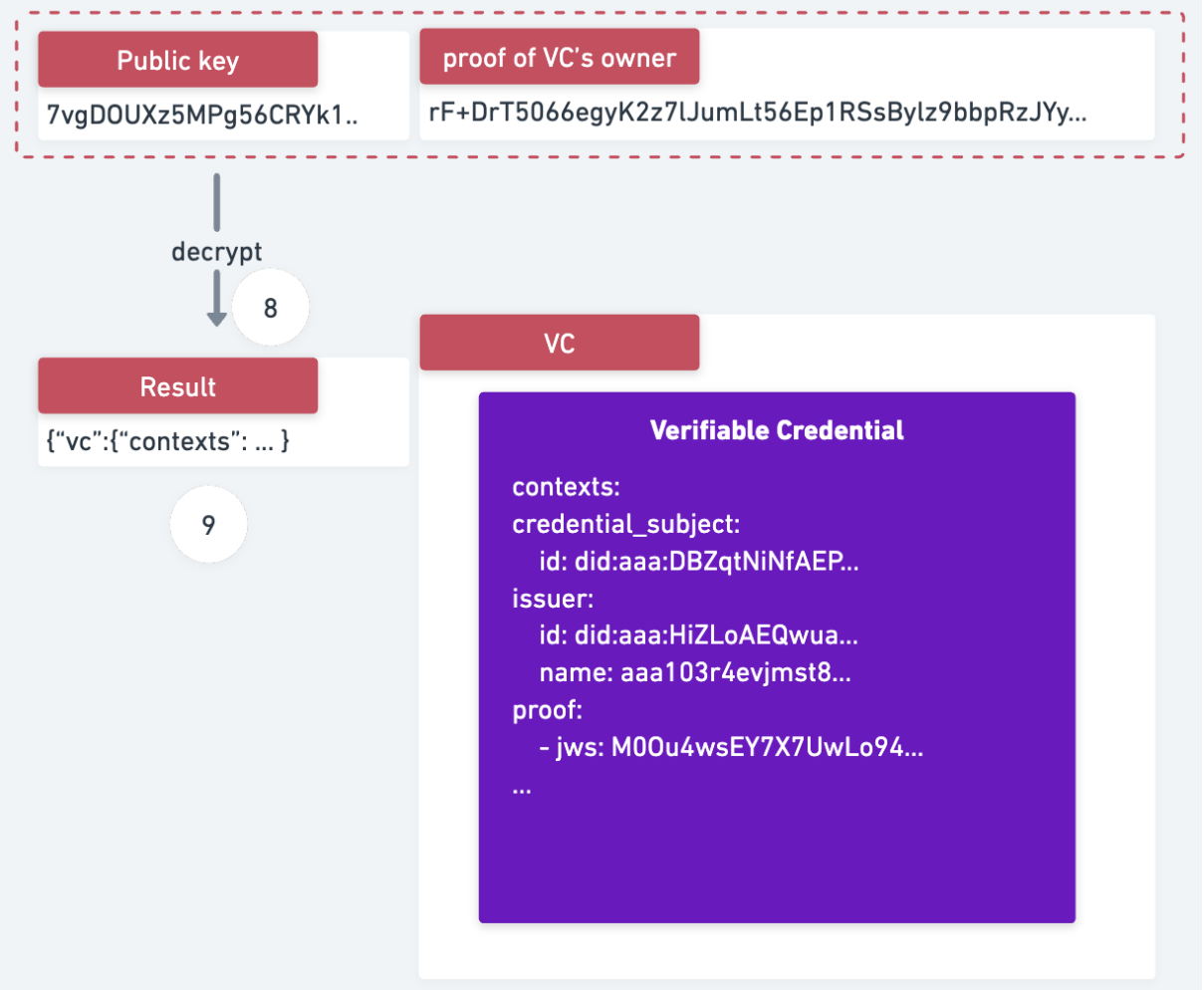



3. VC내에서 발급 대상인 holder를 지칭하는 subject DID 추출
4. 추출한 DID를 base64로 변환(구현채 별로 다름)

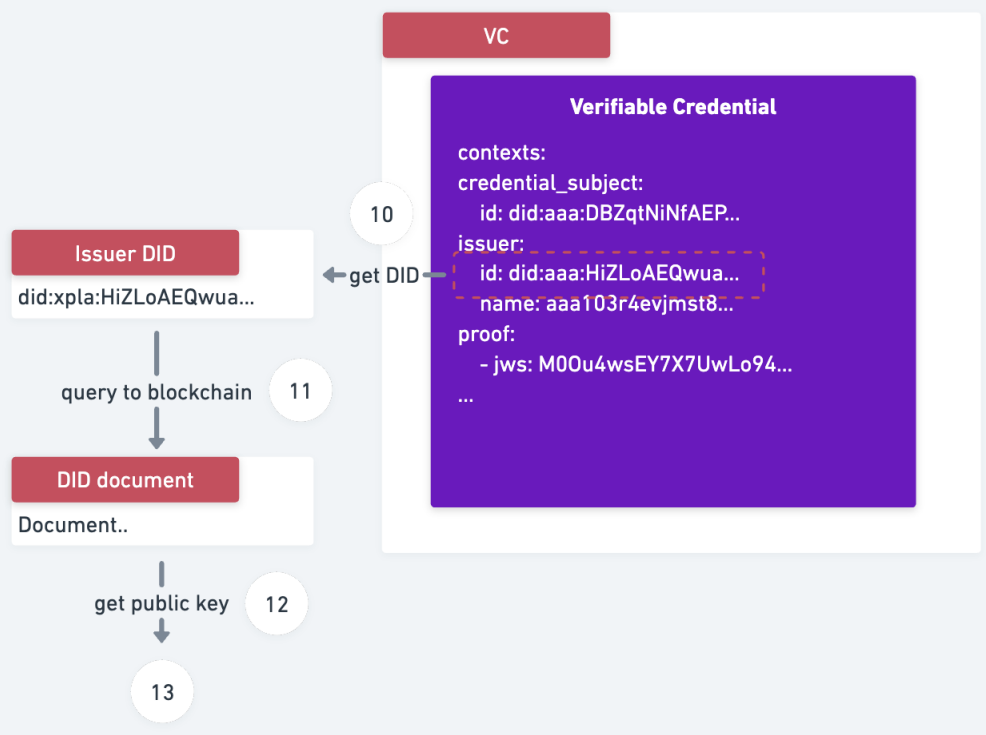

5. 3.에서 추출한 DID와 2.에서 파싱한 method key를 이용하여 DID-method key(verification method)를 생성한다.




6. 4.에서 확인한 DID를 이용하여 블록체인에 기록되어 있는 DID document를 조회한다.
7. DID document 상에서 5.에서 생성한 verification method가 어떤 public key를 가지고 있는지 확인하여 추출한다.




8. 2.에서 파싱한 VP proof를 7.에서 획득한 public key를 이용해서 복호화한다.
9. 복호화시 해당 VP가 결과로 확인되면, VP를 전송한 사람이 holder라는 것을 DID public key를 통해서 검증이 가능하다.




10. Holder 검증이후 VC를 발급한 사람이 올바른 issuer임을 검증하기 위해 VC에서 issuer의 DID를 확인한다.
11. Issuer의 DID를 위 흐름에 따라서 검증한다.

## 실습 코드

1. 전체 코드

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity 0.8.10;

contract Diploma{
    address private issuerAddress;
    uint256 private idCount;
    mapping(uint8 => string) private deptType;

    struct Credential{
        // ID
        uint256 id;
        // 발급자
        address issuer;
        // 학과
```

```
        uint8 deptType;
        // 졸업날짜
        string date;
    }

    mapping(address => Credential) private credentials;

    constructor() {
        issuerAddress = msg.sender;
        idCount = 1;
        deptType[0] = "Computer Sience";
        deptType[1] = "Software";
        deptType[2] = "Mathmaticswjs";
        deptType[3] = "Electronics";
    }

    function claimCredential(address _alumniAddress, uint8
_deptType, string calldata _date) public returns(bool){
        require(issuerAddress == msg.sender, "Not Issuer");
                Credential storage credential = credentials
[_alumniAddress];
        require(credential.id == 0);
        credential.id = idCount;
        credential.issuer = msg.sender;
        credential.deptType = _deptType;
        credential.date = _date;

        idCount += 1;

        return true;
    }

    function hash(uint256 _id, address _issuer, uint8 _dept
Type, string memory _date) pure internal returns(bytes32) {
    return keccak256(abi.encodePacked(_id,_issuer,_deptTyp
e,_date));
    }
```

```
    function getCredential(address _alumniAddress) public v
iew returns (Credential memory){
        return credentials[_alumniAddress];
    }

}
```

## Contract

```
contract Diploma{
    address private issuerAddress;
    uint256 private idCount;
    mapping(uint8 => string) private deptType;

    struct Credential{
        // ID
        uint256 id;
        // 발급자
        address issuer;
        // 학과
        uint8 deptType;
        // 졸업날짜
        string date;
    }
```

VC를 구현하기 위한 구조체

1. id : index 순서를 표기하는 idCount
2. issuer : 발급자
3. deptType : 학과
4. date : 졸업일. VC에 포함되어야 하는 암호화된 정보, 신원, 신원제공자, 엔티티, 서명등으로 JSON형태로 저장된다.

## mapping & constructor

```
    mapping(address => Credential) private credentials;
```

```
    constructor() {
        issuerAddress = msg.sender;
        idCount = 1;
        deptType[0] = "Computer Sience";
        deptType[1] = "Software";
        deptType[2] = "Mathmaticswjs";
        deptType[3] = "Electronics";
    }
```

VC 발행될때 사용하는 변수를 맵팽, constructor()에 학과 정보가 기입되어 있음.

## claimCredential

```
    function claimCredential(address _alumniAddress, uint8
_armyType, string calldata _date) public returns(bool){
        require(issuerAddress == msg.sender, "Not Issuer");
                Credential storage credential = credentials
[_alumniAddress];
        require(credential.id == 0);
        credential.id = idCount;
        credential.issuer = msg.sender;
        credential.armyType = _armyType;
        credential.date = _date;

        idCount += 1;

        return true;
    }
```

해당 함수를 통해서 issuer은 holder에게 크리덴셜(Credential)을 발행 가능

## hash, getCredential

```
    function hash(uint256 _id, address _issuer, uint8 _dept
Type, string memory _date) pure internal returns(bytes32) {
    return keccak256(abi.encodePacked(_id,_issuer,_deptTyp
e,_date));
    }
```

```
    function getCredential(address _alumniAddress) public v
iew returns (Credential memory){
        return credentials[_alumniAddress];
    }

}
```

hash 함수는 크리덴셜 값들을 인코딩하고, 해싱을 하기 위해 함수
getCredential 함수는 holder를 통하여 발행한 크리덴셜을 확인할 수 있다