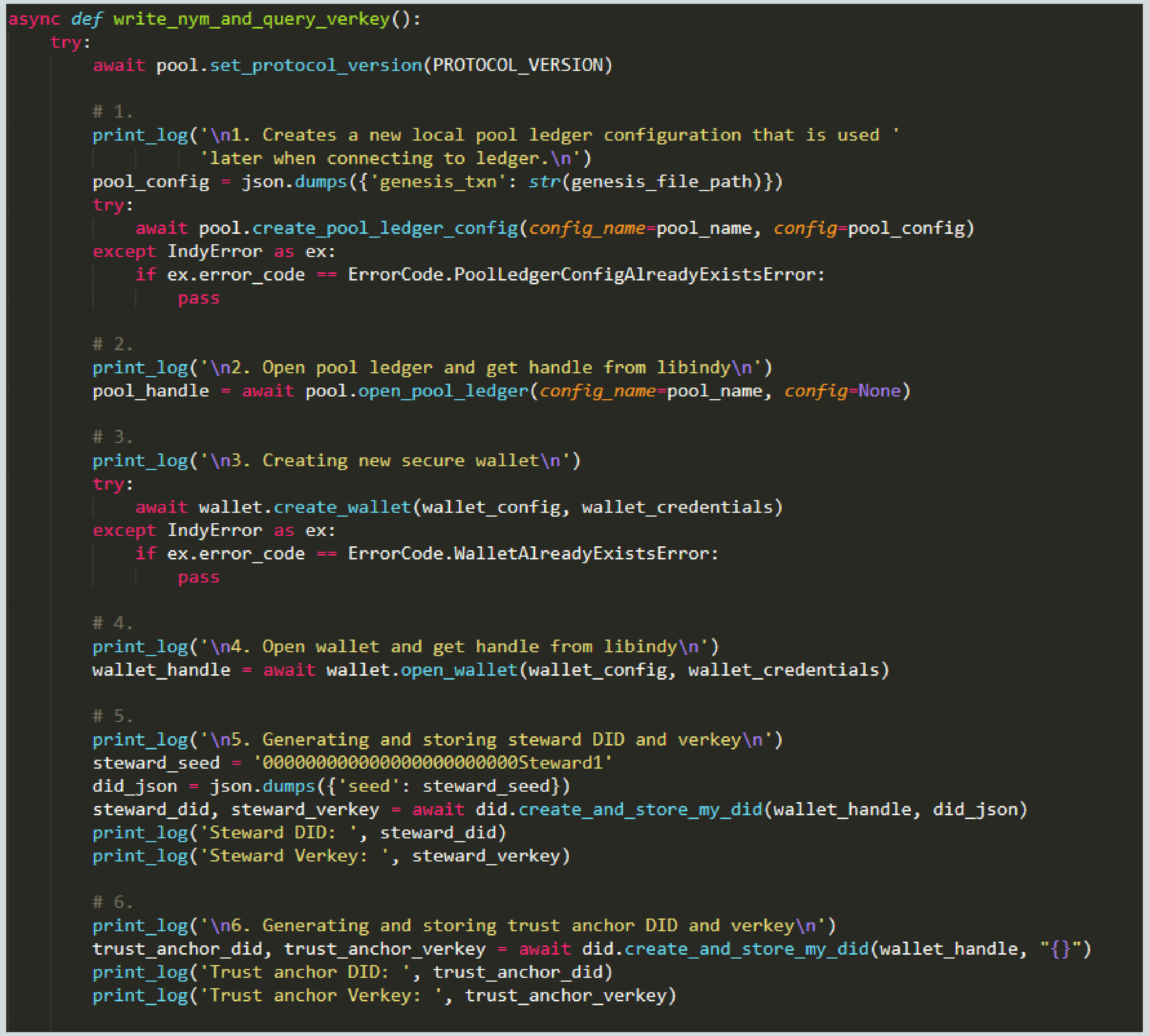
Indy-sdk

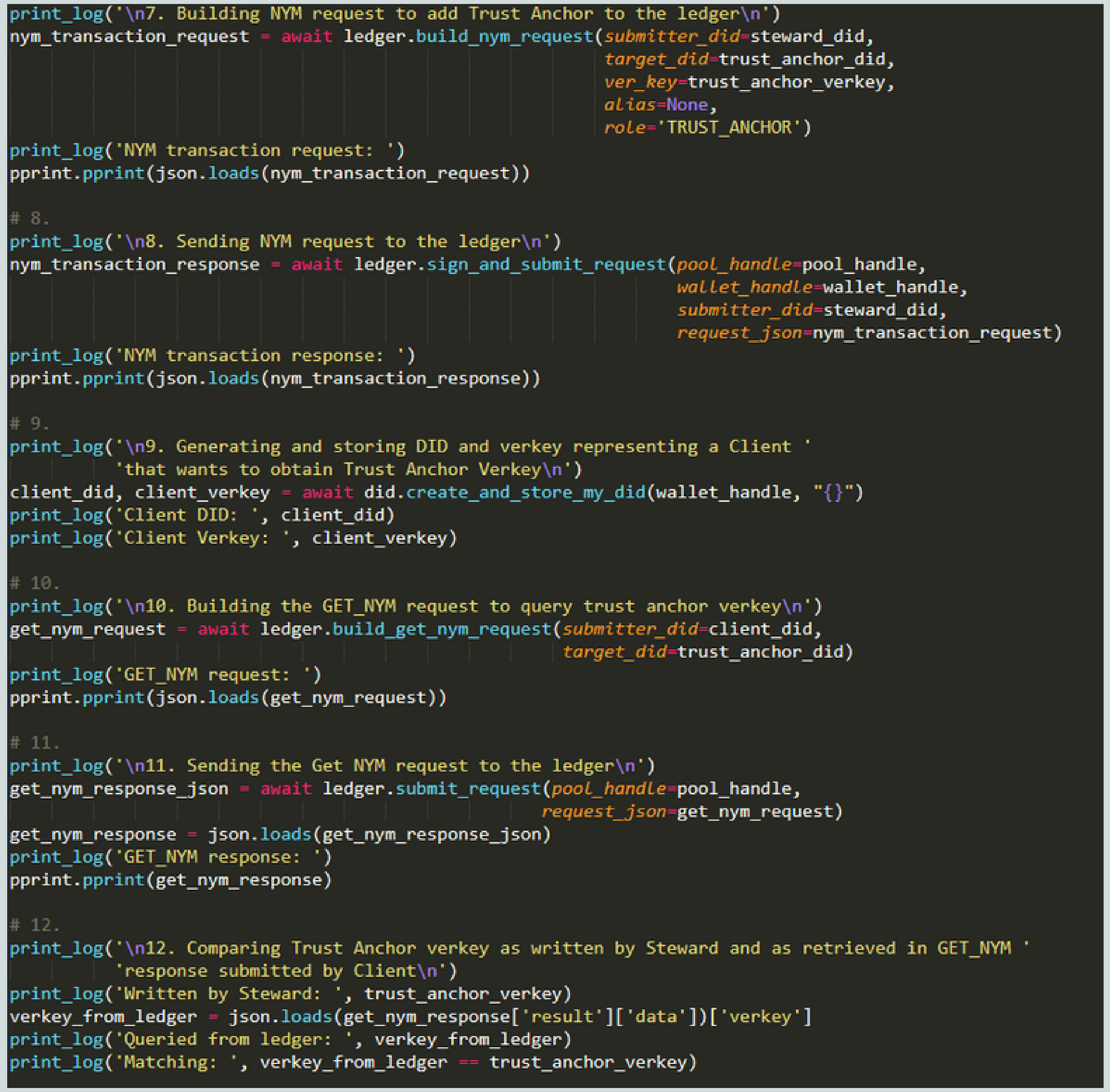
우석대학교 임태희

1. Write a DID and Query Its Verkey



```python
async def write_nym_and_query_verkey():
    try:
        await pool.set_protocol_version(PROTOCOL_VERSION)

        # 1.
        print_log('\n1. Creates a new local pool ledger configuration that is used '
                  'later when connecting to ledger.\n')
        pool_config = json.dumps({'genesis_txn': str(genesis_file_path)})
        try:
            await pool.create_pool_ledger_config(config_name=pool_name, config=pool_config)
        except IndyError as ex:
            if ex.error_code == ErrorCode.PoolLedgerConfigAlreadyExistsError:
                pass

        # 2.
        print_log('\n2. Open pool ledger and get handle from libindy\n')
        pool_handle = await pool.open_pool_ledger(config_name=pool_name, config=None)

        # 3.
        print_log('\n3. Creating new secure wallet\n')
        try:
            await wallet.create_wallet(wallet_config, wallet_credentials)
        except IndyError as ex:
            if ex.error_code == ErrorCode.WalletAlreadyExistsError:
                pass

        # 4.
        print_log('\n4. Open wallet and get handle from libindy\n')
        wallet_handle = await wallet.open_wallet(wallet_config, wallet_credentials)

        # 5.
        print_log('\n5. Generating and storing steward DID and verkey\n')
        steward_seed = '000000000000000000000000Steward1'
        did_json = json.dumps({'seed': steward_seed})
        steward_did, steward_verkey = await did.create_and_store_my_did(wallet_handle, did_json)
        print_log('Steward DID: ', steward_did)
        print_log('Steward Verkey: ', steward_verkey)

        # 6.
        print_log('\n6. Generating and storing trust anchor DID and verkey\n')
        trust_anchor_did, trust_anchor_verkey = await did.create_and_store_my_did(wallet_handle, "{}")
        print_log('Trust anchor DID: ', trust_anchor_did)
        print_log('Trust anchor Verkey: ', trust_anchor_verkey)
```

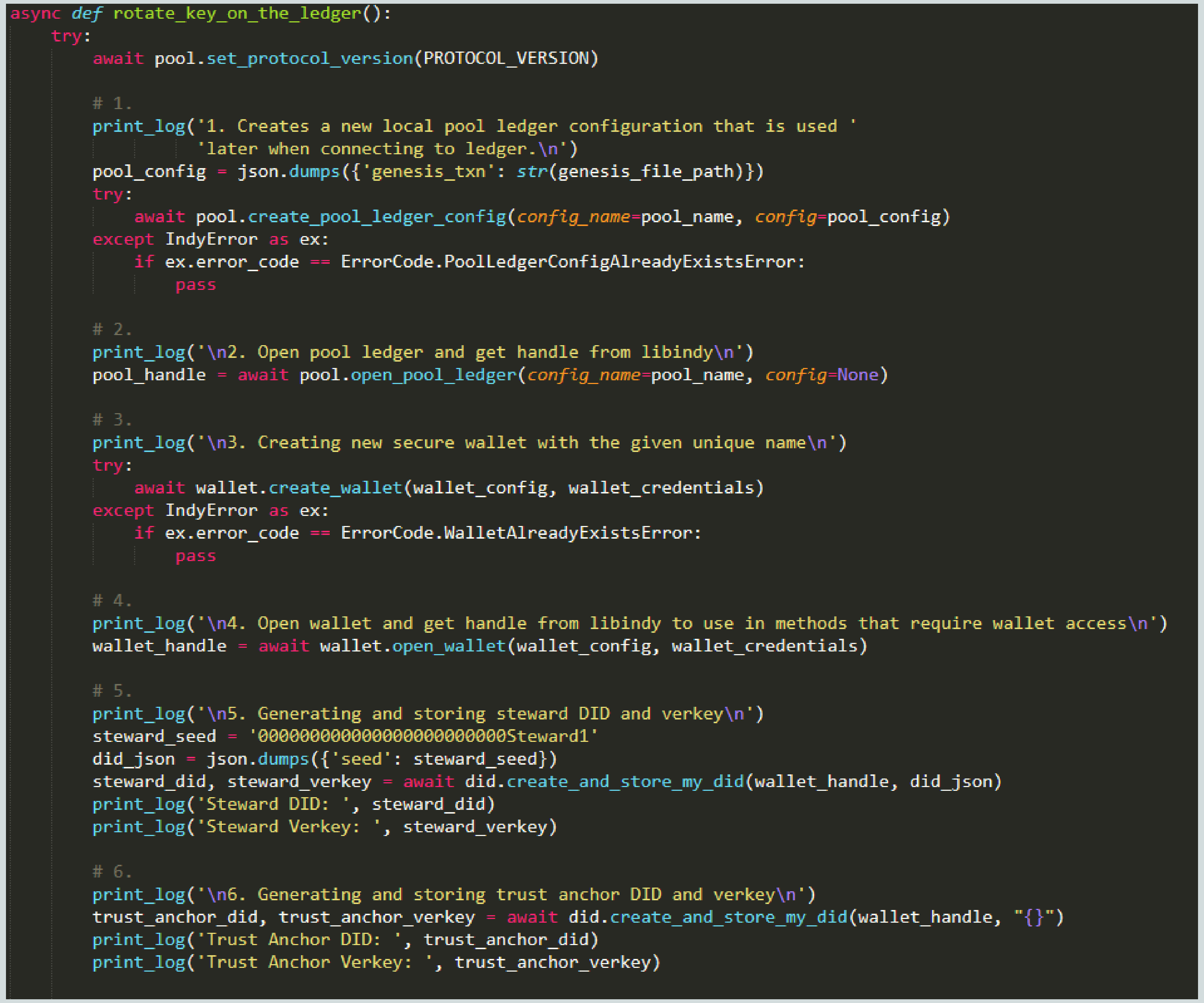
- ledger 연결 설정 및 indy lib연결
- 지갑 생성, 연동 후 DID 저장



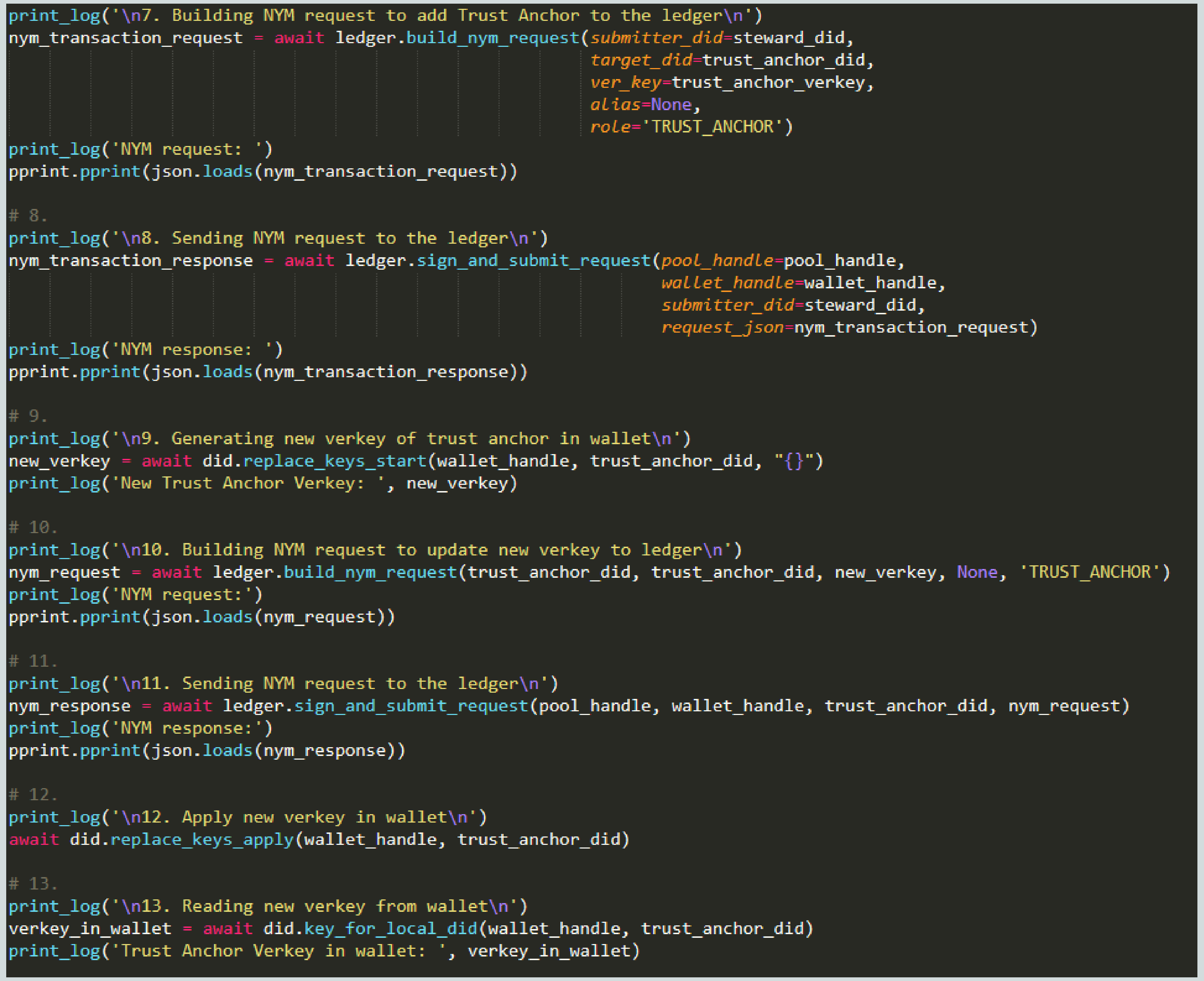
```python
print_log('\n7. Building NYM request to add Trust Anchor to the ledger\n')
nym_transaction_request = await ledger.build_nym_request(submitter_did=steward_did,
                                                         target_did=trust_anchor_did,
                                                         ver_key=trust_anchor_verkey,
                                                         alias=None,
                                                         role='TRUST_ANCHOR')
print_log('NYM transaction request: ')
pprint.pprint(json.loads(nym_transaction_request))

# 8.
print_log('\n8. Sending NYM request to the ledger\n')
nym_transaction_response = await ledger.sign_and_submit_request(pool_handle=pool_handle,
                                                                wallet_handle=wallet_handle,
                                                                submitter_did=steward_did,
                                                                request_json=nym_transaction_request)
print_log('NYM transaction response: ')
pprint.pprint(json.loads(nym_transaction_response))

# 9.
print_log('\n9. Generating and storing DID and verkey representing a Client '
          'that wants to obtain Trust Anchor Verkey\n')
client_did, client_verkey = await did.create_and_store_my_did(wallet_handle, "{}")
print_log('Client DID: ', client_did)
print_log('Client Verkey: ', client_verkey)

# 10.
print_log('\n10. Building the GET_NYM request to query trust anchor verkey\n')
get_nym_request = await ledger.build_get_nym_request(submitter_did=client_did,
                                                     target_did=trust_anchor_did)
print_log('GET_NYM request: ')
pprint.pprint(json.loads(get_nym_request))

# 11.
print_log('\n11. Sending the Get NYM request to the ledger\n')
get_nym_response_json = await ledger.submit_request(pool_handle=pool_handle,
                                                    request_json=get_nym_request)
get_nym_response = json.loads(get_nym_response_json)
print_log('GET_NYM response: ')
pprint.pprint(get_nym_response)

# 12.
print_log('\n12. Comparing Trust Anchor verkey as written by Steward and as retrieved in GET_NYM '
          'response submitted by Client\n')
print_log('Written by Steward: ', trust_anchor_verkey)
verkey_from_ledger = json.loads(get_nym_response['result']['data'])['verkey']
print_log('Queried from ledger: ', verkey_from_ledger)
print_log('Matching: ', verkey_from_ledger == trust_anchor_verkey)
```

- 웹서비스를 제공하는 NYM build 요청, ledger에 anchor 추가
- DID 저장, 대표 verkey(public key) 설정
- GET_NYM build 요청
- 관리자가 쓴 verkey와 response 값 비교

2. Rotate a Key



```python
async def rotate_key_on_the_ledger():
    try:
        await pool.set_protocol_version(PROTOCOL_VERSION)

        # 1.
        print_log('1. Creates a new local pool ledger configuration that is used '
                  'later when connecting to ledger.\n')
        pool_config = json.dumps({'genesis_txn': str(genesis_file_path)})
        try:
            await pool.create_pool_ledger_config(config_name=pool_name, config=pool_config)
        except IndyError as ex:
            if ex.error_code == ErrorCode.PoolLedgerConfigAlreadyExistsError:
                pass

        # 2.
        print_log('\n2. Open pool ledger and get handle from libindy\n')
        pool_handle = await pool.open_pool_ledger(config_name=pool_name, config=None)

        # 3.
        print_log('\n3. Creating new secure wallet with the given unique name\n')
        try:
            await wallet.create_wallet(wallet_config, wallet_credentials)
        except IndyError as ex:
            if ex.error_code == ErrorCode.WalletAlreadyExistsError:
                pass

        # 4.
        print_log('\n4. Open wallet and get handle from libindy to use in methods that require wallet access\n')
        wallet_handle = await wallet.open_wallet(wallet_config, wallet_credentials)

        # 5.
        print_log('\n5. Generating and storing steward DID and verkey\n')
        steward_seed = '000000000000000000000000Steward1'
        did_json = json.dumps({'seed': steward_seed})
        steward_did, steward_verkey = await did.create_and_store_my_did(wallet_handle, did_json)
        print_log('Steward DID: ', steward_did)
        print_log('Steward Verkey: ', steward_verkey)

        # 6.
        print_log('\n6. Generating and storing trust anchor DID and verkey\n')
        trust_anchor_did, trust_anchor_verkey = await did.create_and_store_my_did(wallet_handle, "{}")
        print_log('Trust Anchor DID: ', trust_anchor_did)
        print_log('Trust Anchor Verkey: ', trust_anchor_verkey)
```

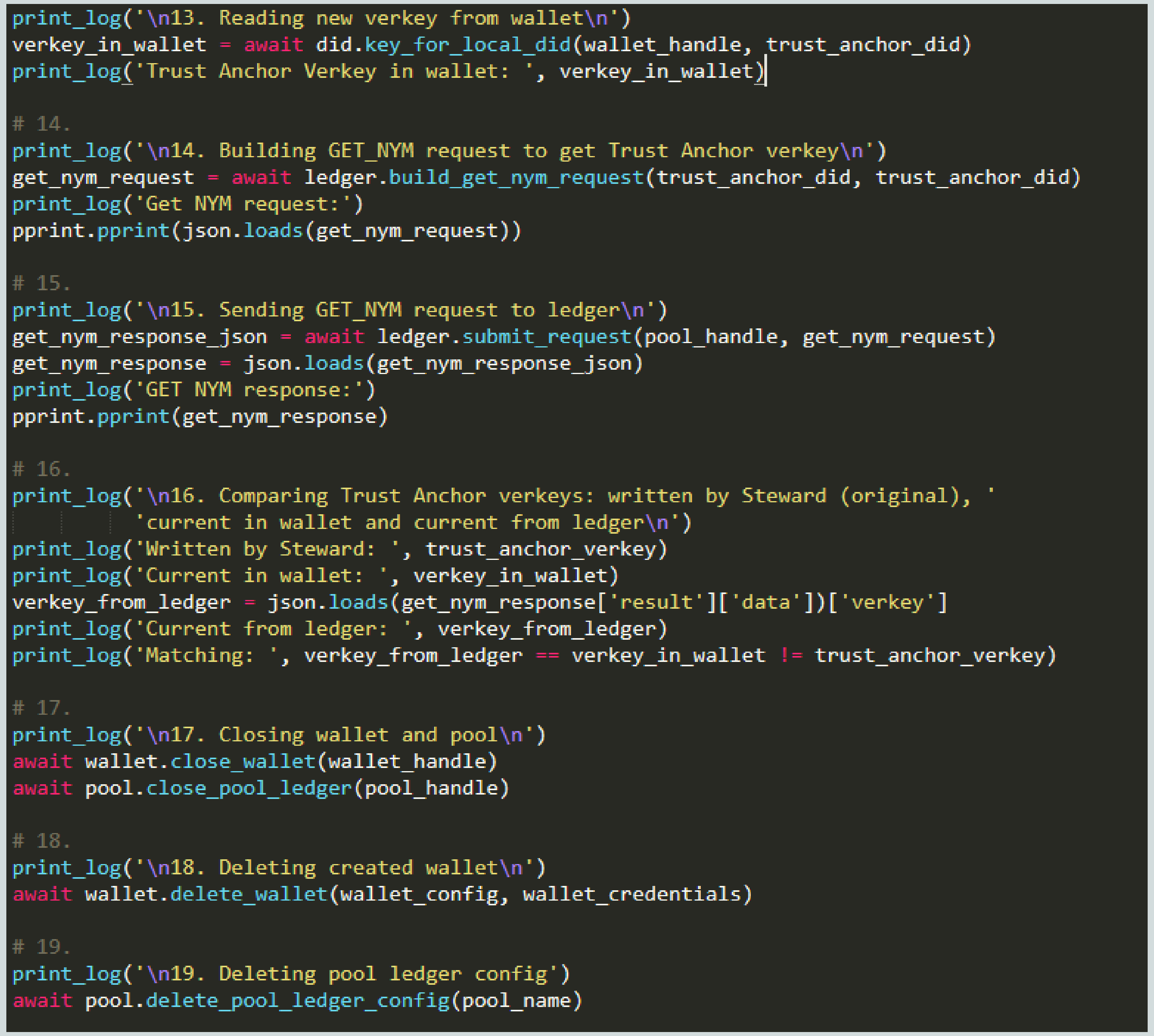


```python
print_log('\n7. Building NYM request to add Trust Anchor to the ledger\n')
nym_transaction_request = await ledger.build_nym_request(submitter_did=steward_did,
                                                         target_did=trust_anchor_did,
                                                         ver_key=trust_anchor_verkey,
                                                         alias=None,
                                                         role='TRUST_ANCHOR')
print_log('NYM request: ')
pprint.pprint(json.loads(nym_transaction_request))

# 8.
print_log('\n8. Sending NYM request to the ledger\n')
nym_transaction_response = await ledger.sign_and_submit_request(pool_handle=pool_handle,
                                                                wallet_handle=wallet_handle,
                                                                submitter_did=steward_did,
                                                                request_json=nym_transaction_request)
print_log('NYM response: ')
pprint.pprint(json.loads(nym_transaction_response))

# 9.
print_log('\n9. Generating new verkey of trust anchor in wallet\n')
new_verkey = await did.replace_keys_start(wallet_handle, trust_anchor_did, "{}")
print_log('New Trust Anchor Verkey: ', new_verkey)

# 10.
print_log('\n10. Building NYM request to update new verkey to ledger\n')
nym_request = await ledger.build_nym_request(trust_anchor_did, trust_anchor_did, new_verkey, None, 'TRUST_ANCHOR')
print_log('NYM request:')
pprint.pprint(json.loads(nym_request))

# 11.
print_log('\n11. Sending NYM request to the ledger\n')
nym_response = await ledger.sign_and_submit_request(pool_handle, wallet_handle, trust_anchor_did, nym_request)
print_log('NYM response:')
pprint.pprint(json.loads(nym_response))

# 12.
print_log('\n12. Apply new verkey in wallet\n')
await did.replace_keys_apply(wallet_handle, trust_anchor_did)

# 13.
print_log('\n13. Reading new verkey from wallet\n')
verkey_in_wallet = await did.key_for_local_did(wallet_handle, trust_anchor_did)
print_log('Trust Anchor Verkey in wallet: ', verkey_in_wallet)
```

- 추후 다시 연결할 장부 생성 및 설정
- 지갑 생성, 지갑에 접근하기 위한 매소드 설정
- 관리자 DID, verkey 생성 및 저장

- NYM 요청 build, 관리자에 Trust Anchor 추가
- 관리자에 NYM 전송 요청
- Trust Anchor에 대한 새로운 verkey를 지갑에 생성
- 새로운 verkey를 지갑에 적용 후 읽음

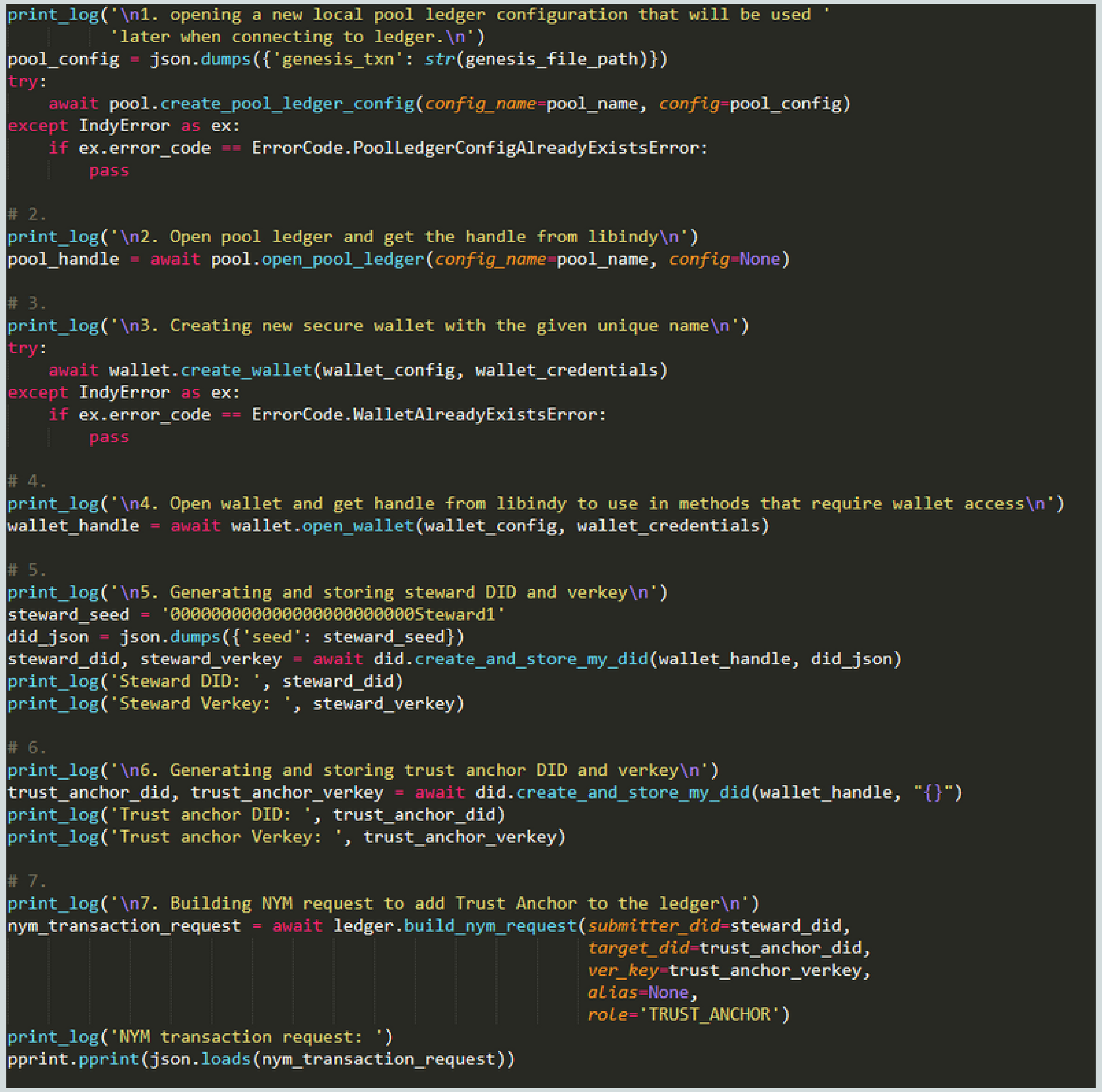
2. Rotate a Key



```python
print_log('\n13. Reading new verkey from wallet\n')
verkey_in_wallet = await did.key_for_local_did(wallet_handle, trust_anchor_did)
print_log('Trust Anchor Verkey in wallet: ', verkey_in_wallet)

# 14.
print_log('\n14. Building GET_NYM request to get Trust Anchor verkey\n')
get_nym_request = await ledger.build_get_nym_request(trust_anchor_did, trust_anchor_did)
print_log('Get NYM request:')
pprint.pprint(json.loads(get_nym_request))

# 15.
print_log('\n15. Sending GET_NYM request to ledger\n')
get_nym_response_json = await ledger.submit_request(pool_handle, get_nym_request)
get_nym_response = json.loads(get_nym_response_json)
print_log('GET NYM response:')
pprint.pprint(get_nym_response)

# 16.
print_log('\n16. Comparing Trust Anchor verkeys: written by Steward (original), '
          'current in wallet and current from ledger\n')
print_log('Written by Steward: ', trust_anchor_verkey)
print_log('Current in wallet: ', verkey_in_wallet)
verkey_from_ledger = json.loads(get_nym_response['result']['data'])['verkey']
print_log('Current from ledger: ', verkey_from_ledger)
print_log('Matching: ', verkey_from_ledger == verkey_in_wallet != trust_anchor_verkey)

# 17.
print_log('\n17. Closing wallet and pool\n')
await wallet.close_wallet(wallet_handle)
await pool.close_pool_ledger(pool_handle)

# 18.
print_log('\n18. Deleting created wallet\n')
await wallet.delete_wallet(wallet_config, wallet_credentials)

# 19.
print_log('\n19. Deleting pool ledger config')
await pool.delete_pool_ledger_config(pool_name)
```

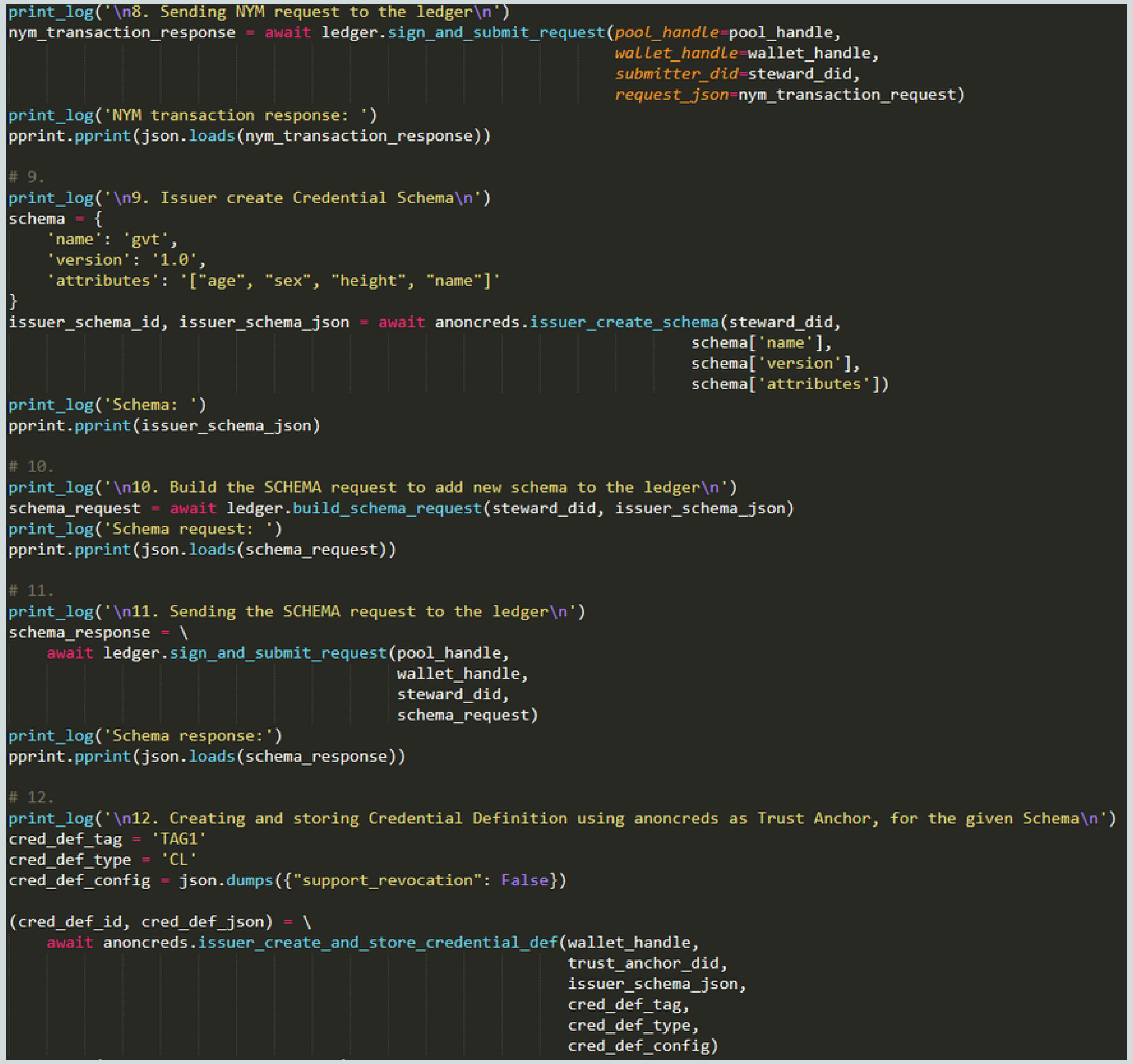
- Trust Anchor verkey를 얻기 위한 GET_NYM 요청 build
- 관리자에 GET_NYM 전송 요청
- 관리자가 쓴것과 장부와 비교
- 지갑 close, delete, 장부 설정 삭제

3. Save a Schema and Cred Def



```python
print_log('\n1. opening a new local pool ledger configuration that will be used '
            'later when connecting to ledger.\n')
pool_config = json.dumps({'genesis_txn': str(genesis_file_path)})
try:
    await pool.create_pool_ledger_config(config_name=pool_name, config=pool_config)
except IndyError as ex:
    if ex.error_code == ErrorCode.PoolLedgerConfigAlreadyExistsError:
        pass

# 2.
print_log('\n2. Open pool ledger and get the handle from libindy\n')
pool_handle = await pool.open_pool_ledger(config_name=pool_name, config=None)

# 3.
print_log('\n3. Creating new secure wallet with the given unique name\n')
try:
    await wallet.create_wallet(wallet_config, wallet_credentials)
except IndyError as ex:
    if ex.error_code == ErrorCode.WalletAlreadyExistsError:
        pass

# 4.
print_log('\n4. Open wallet and get handle from libindy to use in methods that require wallet access\n')
wallet_handle = await wallet.open_wallet(wallet_config, wallet_credentials)

# 5.
print_log('\n5. Generating and storing steward DID and verkey\n')
steward_seed = '000000000000000000000000Steward1'
did_json = json.dumps({'seed': steward_seed})
steward_did, steward_verkey = await did.create_and_store_my_did(wallet_handle, did_json)
print_log('Steward DID: ', steward_did)
print_log('Steward Verkey: ', steward_verkey)

# 6.
print_log('\n6. Generating and storing trust anchor DID and verkey\n')
trust_anchor_did, trust_anchor_verkey = await did.create_and_store_my_did(wallet_handle, "{}")
print_log('Trust anchor DID: ', trust_anchor_did)
print_log('Trust anchor Verkey: ', trust_anchor_verkey)

# 7.
print_log('\n7. Building NYM request to add Trust Anchor to the ledger\n')
nym_transaction_request = await ledger.build_nym_request(submitter_did=steward_did,
                                                         target_did=trust_anchor_did,
                                                         ver_key=trust_anchor_verkey,
                                                         alias=None,
                                                         role='TRUST_ANCHOR')
print_log('NYM transaction request: ')
pprint.pprint(json.loads(nym_transaction_request))
```

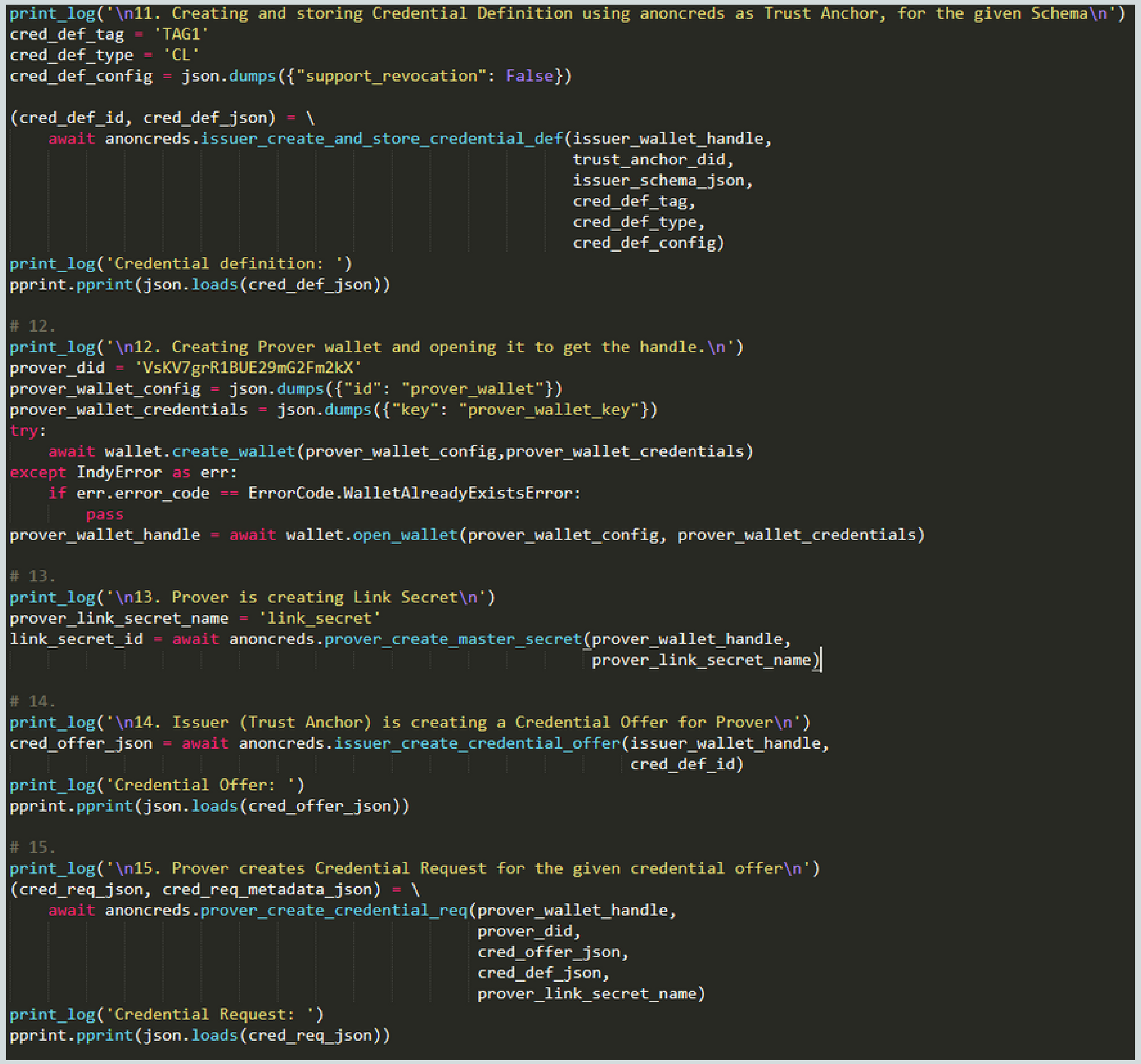
- 추후 연결 할 수도 있는 장부 설정
- 지갑 생성 및 접근을 위한 매소드
- 관리자 DID, verkey를 생성 및 저장
- Trust Anchor 장부에 추가할 NYM 요청 build



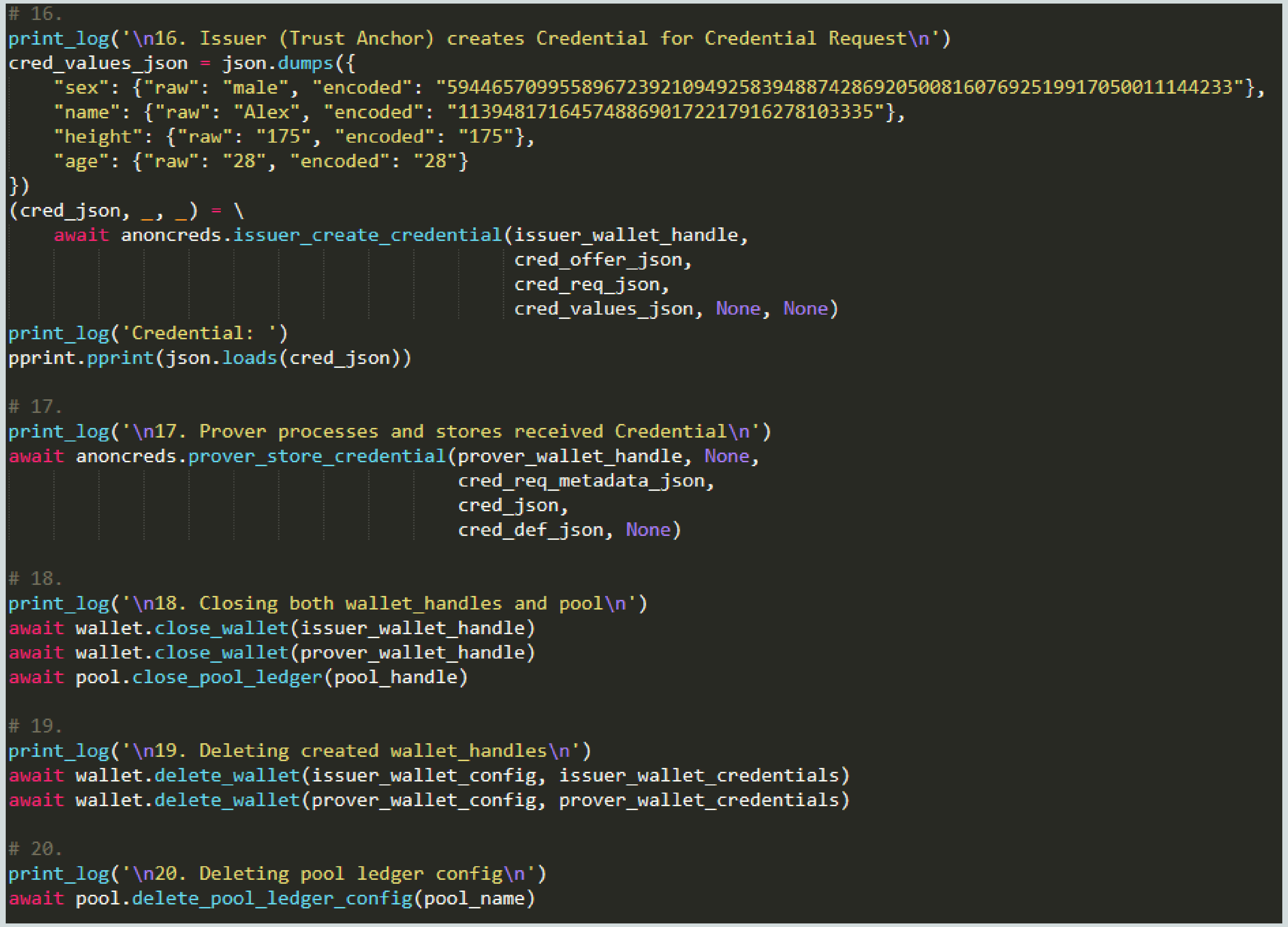
```python
print_log('\n8. Sending NYM request to the ledger\n')
nym_transaction_response = await ledger.sign_and_submit_request(pool_handle=pool_handle,
                                                                wallet_handle=wallet_handle,
                                                                submitter_did=steward_did,
                                                                request_json=nym_transaction_request)
print_log('NYM transaction response: ')
pprint.pprint(json.loads(nym_transaction_response))

# 9.
print_log('\n9. Issuer create Credential Schema\n')
schema = {
    'name': 'gvt',
    'version': '1.0',
    'attributes': '["age", "sex", "height", "name"]'
}
issuer_schema_id, issuer_schema_json = await anoncreds.issuer_create_schema(steward_did,
                                                                            schema['name'],
                                                                            schema['version'],
                                                                            schema['attributes'])
print_log('Schema: ')
pprint.pprint(issuer_schema_json)

# 10.
print_log('\n10. Build the SCHEMA request to add new schema to the ledger\n')
schema_request = await ledger.build_schema_request(steward_did, issuer_schema_json)
print_log('Schema request: ')
pprint.pprint(json.loads(schema_request))

# 11.
print_log('\n11. Sending the SCHEMA request to the ledger\n')
schema_response = \
    await ledger.sign_and_submit_request(pool_handle,
                                         wallet_handle,
                                         steward_did,
                                         schema_request)
print_log('Schema response:')
pprint.pprint(json.loads(schema_response))

# 12.
print_log('\n12. Creating and storing Credential Definition using anoncreds as Trust Anchor, for the given Schema\n')
cred_def_tag = 'TAG1'
cred_def_type = 'CL'
cred_def_config = json.dumps({"support_revocation": False})

(cred_def_id, cred_def_json) = \
    await anoncreds.issuer_create_and_store_credential_def(wallet_handle,
                                                           trust_anchor_did,
                                                           issuer_schema_json,
                                                           cred_def_tag,
                                                           cred_def_type,
                                                           cred_def_config)
```

- 장부에 NYM 전송 요청
- 발행인이 증명 스키마 생성
- 장부에 schema 요청 전송
- schema에 대해 익명을 Trust Anchor로 사용하여 자격 증명 정의 생성 및 저장

4. Issue a Credential



```python
print_log('\n11. Creating and storing Credential Definition using anoncreds as Trust Anchor, for the given Schema\n')
cred_def_tag = 'TAG1'
cred_def_type = 'CL'
cred_def_config = json.dumps({"support_revocation": False})

(cred_def_id, cred_def_json) = \
    await anoncreds.issuer_create_and_store_credential_def(issuer_wallet_handle,
                                                           trust_anchor_did,
                                                           issuer_schema_json,
                                                           cred_def_tag,
                                                           cred_def_type,
                                                           cred_def_config)
print_log('Credential definition: ')
pprint.pprint(json.loads(cred_def_json))

# 12.
print_log('\n12. Creating Prover wallet and opening it to get the handle.\n')
prover_did = 'VsKV7grR1BUE29mG2Fm2kX'
prover_wallet_config = json.dumps({"id": "prover_wallet"})
prover_wallet_credentials = json.dumps({"key": "prover_wallet_key"})
try:
    await wallet.create_wallet(prover_wallet_config,prover_wallet_credentials)
except IndyError as err:
    if err.error_code == ErrorCode.WalletAlreadyExistsError:
        pass
prover_wallet_handle = await wallet.open_wallet(prover_wallet_config, prover_wallet_credentials)

# 13.
print_log('\n13. Prover is creating Link Secret\n')
prover_link_secret_name = 'link_secret'
link_secret_id = await anoncreds.prover_create_master_secret(prover_wallet_handle,
                                                             prover_link_secret_name)

# 14.
print_log('\n14. Issuer (Trust Anchor) is creating a Credential Offer for Prover\n')
cred_offer_json = await anoncreds.issuer_create_credential_offer(issuer_wallet_handle,
                                                                 cred_def_id)
print_log('Credential Offer: ')
pprint.pprint(json.loads(cred_offer_json))

# 15.
print_log('\n15. Prover creates Credential Request for the given credential offer\n')
(cred_req_json, cred_req_metadata_json) = \
    await anoncreds.prover_create_credential_req(prover_wallet_handle,
                                                 prover_did,
                                                 cred_offer_json,
                                                 cred_def_json,
                                                 prover_link_secret_name)
print_log('Credential Request: ')
pprint.pprint(json.loads(cred_req_json))
```

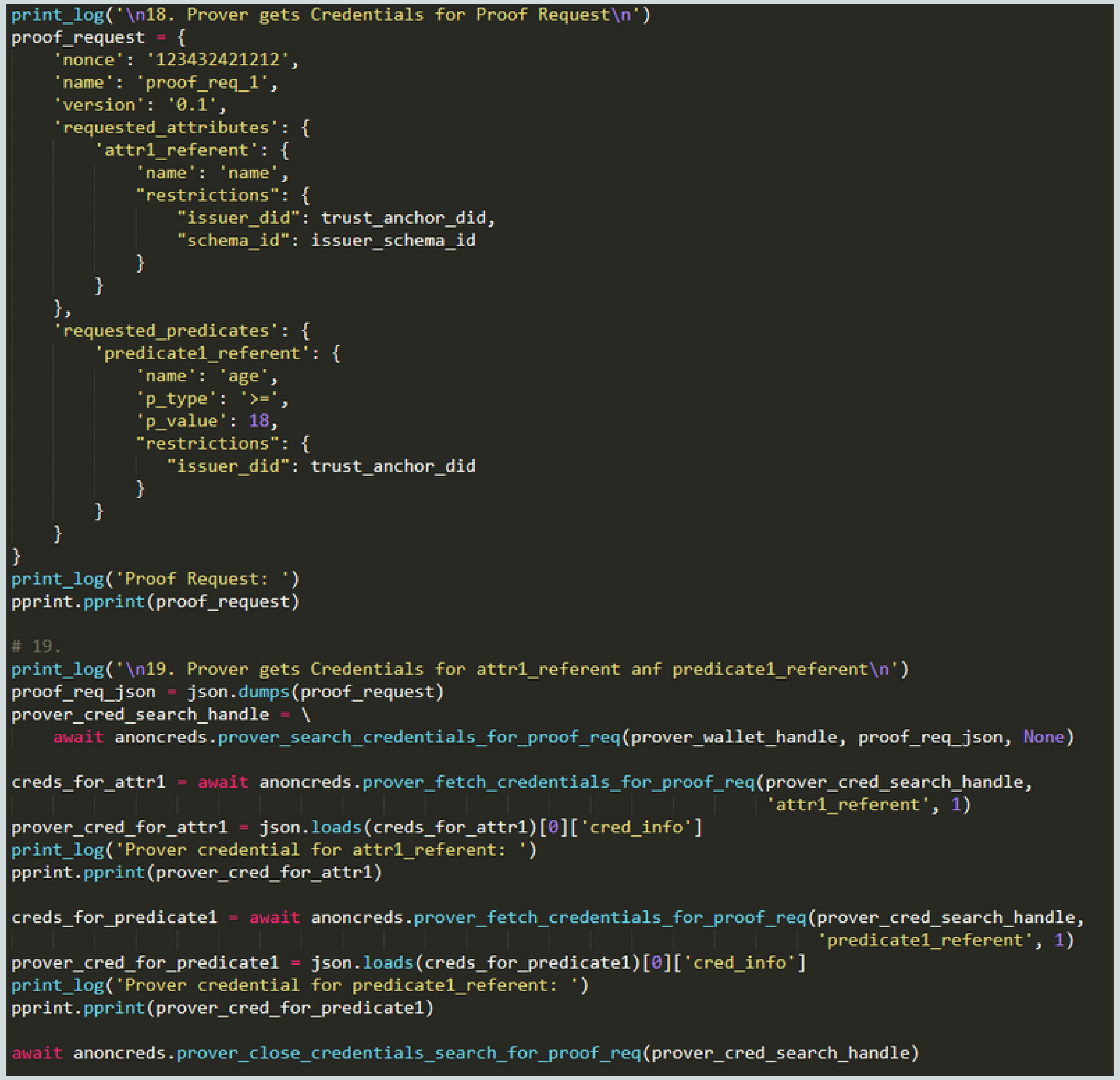


```python
# 16.
print_log('\n16. Issuer (Trust Anchor) creates Credential for Credential Request\n')
cred_values_json = json.dumps({
    "sex": {"raw": "male", "encoded": "5944657099558967239210949258394887428692050081607692519917050011144233"},
    "name": {"raw": "Alex", "encoded": "1139481716457488690172217916278103335"},
    "height": {"raw": "175", "encoded": "175"},
    "age": {"raw": "28", "encoded": "28"}
})
(cred_json, _, _) = \
    await anoncreds.issuer_create_credential(issuer_wallet_handle,
                                             cred_offer_json,
                                             cred_req_json,
                                             cred_values_json, None, None)
print_log('Credential: ')
pprint.pprint(json.loads(cred_json))

# 17.
print_log('\n17. Prover processes and stores received Credential\n')
await anoncreds.prover_store_credential(prover_wallet_handle, None,
                                        cred_req_metadata_json,
                                        cred_json,
                                        cred_def_json, None)

# 18.
print_log('\n18. Closing both wallet_handles and pool\n')
await wallet.close_wallet(issuer_wallet_handle)
await wallet.close_wallet(prover_wallet_handle)
await pool.close_pool_ledger(pool_handle)

# 19.
print_log('\n19. Deleting created wallet_handles\n')
await wallet.delete_wallet(issuer_wallet_config, issuer_wallet_credentials)
await wallet.delete_wallet(prover_wallet_config, prover_wallet_credentials)

# 20.
print_log('\n20. Deleting pool ledger config\n')
await pool.delete_pool_ledger_config(pool_name)
```

- schema에 대해 익명을 Trust Anchor로 사용하여 자격 증명서 생성 및 저장
- 지갑 견본 생성 및 사용
- 견본은 secret link로 생성
- Trust Anchor는 prover를 위한 자격 증명서를 생성

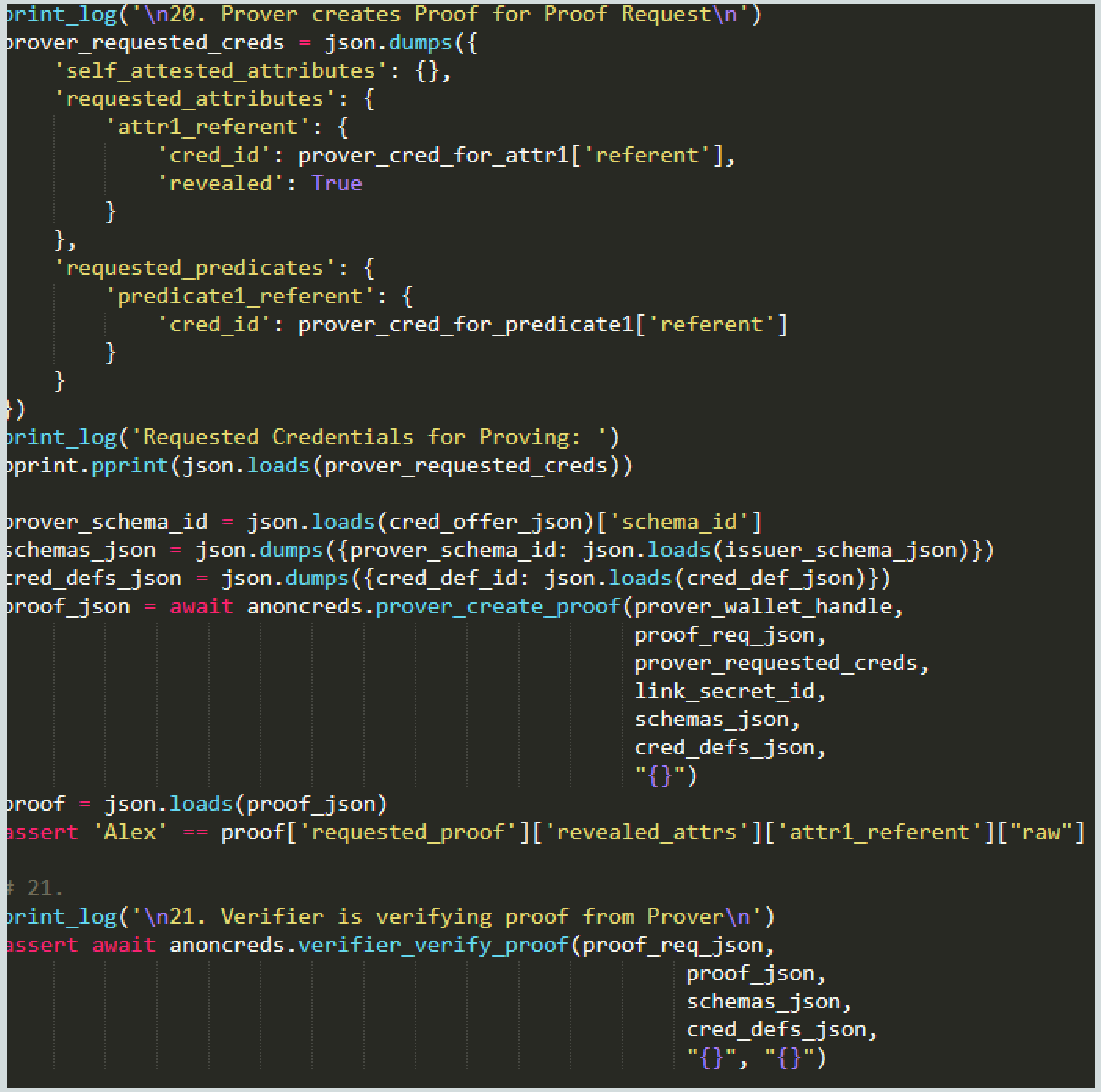
- Trust Anchor는 자격증명 요청에 대한 자격 증명 생성
- 공동 계산 및 지갑 핸들러 삭제, ledger 설정 삭제

5. Negotiate a Proof



```
print_log('\n18. Prover gets Credentials for Proof Request\n')
proof_request = {
    'nonce': '123432421212',
    'name': 'proof_req_1',
    'version': '0.1',
    'requested_attributes': {
        'attr1_referent': {
            'name': 'name',
            "restrictions": {
                "issuer_did": trust_anchor_did,
                "schema_id": issuer_schema_id
            }
        }
    },
    'requested_predicates': {
        'predicate1_referent': {
            'name': 'age',
            'p_type': '>=',
            'p_value': 18,
            "restrictions": {
               "issuer_did": trust_anchor_did
            }
        }
    }
}
print_log('Proof Request: ')
pprint.pprint(proof_request)

# 19.
print_log('\n19. Prover gets Credentials for attr1_referent anf predicate1_referent\n')
proof_req_json = json.dumps(proof_request)
prover_cred_search_handle = \
    await anoncreds.prover_search_credentials_for_proof_req(prover_wallet_handle, proof_req_json, None)

creds_for_attr1 = await anoncreds.prover_fetch_credentials_for_proof_req(prover_cred_search_handle,
                                                                          'attr1_referent', 1)
prover_cred_for_attr1 = json.loads(creds_for_attr1)[0]['cred_info']
print_log('Prover credential for attr1_referent: ')
pprint.pprint(prover_cred_for_attr1)

creds_for_predicate1 = await anoncreds.prover_fetch_credentials_for_proof_req(prover_cred_search_handle,
                                                                               'predicate1_referent', 1)
prover_cred_for_predicate1 = json.loads(creds_for_predicate1)[0]['cred_info']
print_log('Prover credential for predicate1_referent: ')
pprint.pprint(prover_cred_for_predicate1)

await anoncreds.prover_close_credentials_search_for_proof_req(prover_cred_search_handle)
```

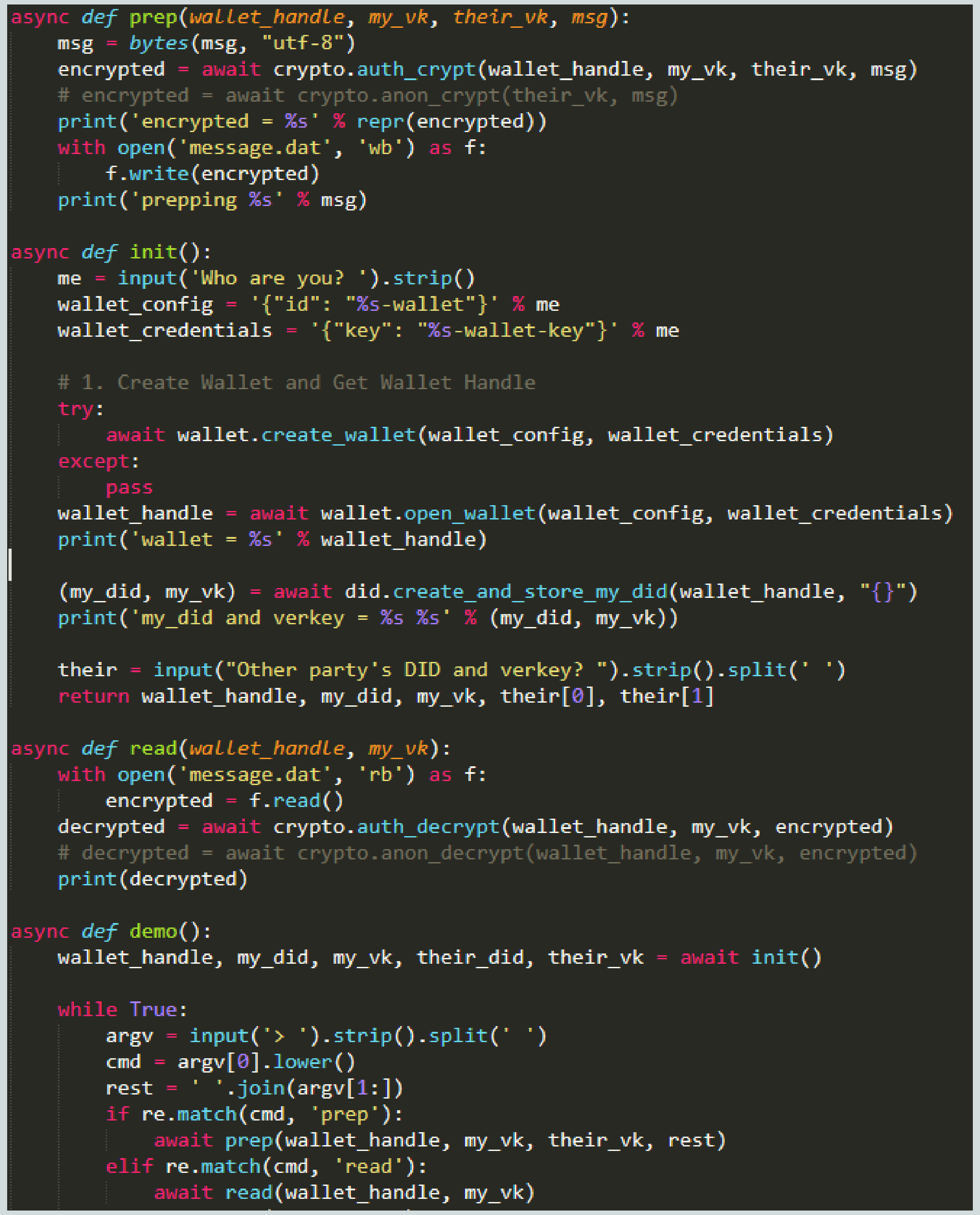
- prover는 증명 요청에 대한 자격증명을 얻음.
- prover는 attr1_referent와 predicate1_referent에 대한 자격증명을 얻음



```
rint_log('\n20. Prover creates Proof for Proof Request\n')
rover_requested_creds = json.dumps({
    'self_attested_attributes': {},
    'requested_attributes': {
        'attr1_referent': {
            'cred_id': prover_cred_for_attr1['referent'],
            'revealed': True
        }
    },
    'requested_predicates': {
        'predicate1_referent': {
            'cred_id': prover_cred_for_predicate1['referent']
        }
    }
})
rint_log('Requested Credentials for Proving: ')
print.pprint(json.loads(prover_requested_creds))

rover_schema_id = json.loads(cred_offer_json)['schema_id']
chemas_json = json.dumps({prover_schema_id: json.loads(issuer_schema_json)})
red_defs_json = json.dumps({cred_def_id: json.loads(cred_def_json)})
roof_json = await anoncreds.prover_create_proof(prover_wallet_handle,
                                                proof_req_json,
                                                prover_requested_creds,
                                                link_secret_id,
                                                schemas_json,
                                                cred_defs_json,
                                                "{}")
roof = json.loads(proof_json)
ssert 'Alex' == proof['requested_proof']['revealed_attrs']['attr1_referent']["raw"]

 21.
rint_log('\n21. Verifier is verifying proof from Prover\n')
ssert await anoncreds.verifier_verify_proof(proof_req_json,
                                            proof_json,
                                            schemas_json,
                                            cred_defs_json,
                                            "{}", "{}")
```

- prover는 요청 증명에 대한 증명서를 생성
- 증명을 위해 요청
- 검증자는 prover의 증명을 검토
- 지갑과 pool을 닫고 지갑 핸들러 삭제

6. Send a Secure Message



```python
async def prep(wallet_handle, my_vk, their_vk, msg):
    msg = bytes(msg, "utf-8")
    encrypted = await crypto.auth_crypt(wallet_handle, my_vk, their_vk, msg)
    # encrypted = await crypto.anon_crypt(their_vk, msg)
    print('encrypted = %s' % repr(encrypted))
    with open('message.dat', 'wb') as f:
        f.write(encrypted)
    print('prepping %s' % msg)

async def init():
    me = input('Who are you? ').strip()
    wallet_config = '{"id": "%s-wallet"}' % me
    wallet_credentials = '{"key": "%s-wallet-key"}' % me

    # 1. Create Wallet and Get Wallet Handle
    try:
        await wallet.create_wallet(wallet_config, wallet_credentials)
    except:
        pass
    wallet_handle = await wallet.open_wallet(wallet_config, wallet_credentials)
    print('wallet = %s' % wallet_handle)

    (my_did, my_vk) = await did.create_and_store_my_did(wallet_handle, "{}")
    print('my_did and verkey = %s %s' % (my_did, my_vk))

    their = input("Other party's DID and verkey? ").strip().split(' ')
    return wallet_handle, my_did, my_vk, their[0], their[1]

async def read(wallet_handle, my_vk):
    with open('message.dat', 'rb') as f:
        encrypted = f.read()
    decrypted = await crypto.auth_decrypt(wallet_handle, my_vk, encrypted)
    # decrypted = await crypto.anon_decrypt(wallet_handle, my_vk, encrypted)
    print(decrypted)

async def demo():
    wallet_handle, my_did, my_vk, their_did, their_vk = await init()

    while True:
        argv = input('> ').strip().split(' ')
        cmd = argv[0].lower()
        rest = ' '.join(argv[1:])
        if re.match(cmd, 'prep'):
            await prep(wallet_handle, my_vk, their_vk, rest)
        elif re.match(cmd, 'read'):
            await read(wallet_handle, my_vk)
```

- id, 키 확인 및 지갑 생성