10주차 – 프로젝트 전체 설계

프로젝트 이름 – 파일기반 인증, 암호화 블록체인 시스템

- 노드, 클라이언트 프로그램 2가지 종류로 구성
- 노드 프로그램을 가지고 암호화 기능과 인증기능을 제공하는 파일기반의 사설 블록체인 망을 직접 구축가능. 보안기능을 제공하기 때문에 외부인의 사용은 불가함.

특징

- 노드가입, 클라이언트 가입은 모든 노드가 승인해야 가능(공개키를 등록하는 과정)
- 노드, 클라이언트등 모든 사설망 안의 통신은 RSA인증을 거쳐 이루어짐(일단 망이 개설되면 외부 간섭이 불가능)

보안대책

1. 클라이언트로 가장하고 계속해서 파일 업로드 요청을 보내는 경우

- 업로드 요청을 하려면 사전에 블록체인 망에 등록한 RSA키가 필요, 인증에 실패하면 해당 요청은 무시됨

2. 노드로 가장하고 계속해서 악의적인 요청을 망에 보내는 경우

- 노드를 망에 추가할 때 RSA 키를 등록하므로 RSA키가 없다면 해당 요청은 무시됨

3. RSA서명 구조

- 클라이언트-노드 간 통신의 경우 서명의 구조는 "망 전체가 공유하는 고유의 문자열" + "클라이언트 Transaction 카운트" 로 이루어진다.

-노드-노드 간 통신의 경우 서명의 구조는 "망 전체가 공유하는 고유의 문자열" + "노드 Transaction 카운트" 로 이루어진다.

- 즉 매 통신마다 +1씩 변화하는 카운트 값을 가지고 서명을 생성하므로 각 서명은 한번만 유효하다. 따라서 인터넷에서 외부인이 서명을 중간에 탈취하더라도 사용할 수 없다.

-Decentralization
-특정기업, 기관이 아니라 모두가 서로의 데이터를 암호화된 형태로 저장,
-분산 저장으로 용량에 이점
-더 많은 백업파일 저장이 가능해짐
-노드 간의 합의에 의해 데이터의 무결성을 보장

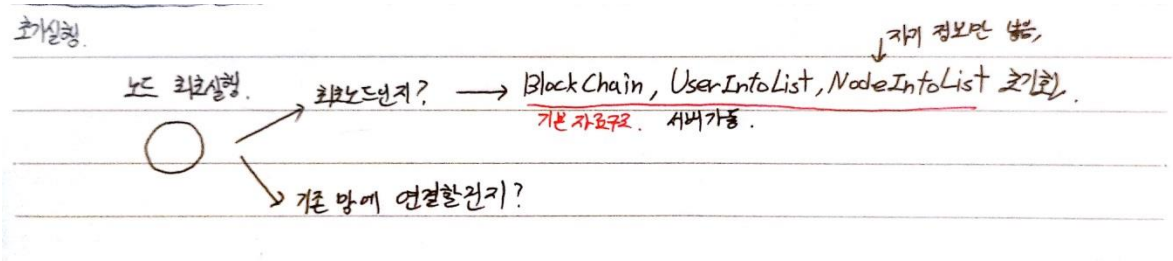
-Anonymity
서로 이 파일이 누구의 파일인지, 내용이 무엇인지 알 수 없음.

-Persistency
-한번 블록체인에 저장된 파일은 무결성을 보장받고 삭제되지 않으며 언제든지 다시 접근가능
-에러발생시대처가능

-Auditability
-가입자의 공개키를 기준 으로 사용자들을 관리, 저장한 파일 등의 정보를 접근가능

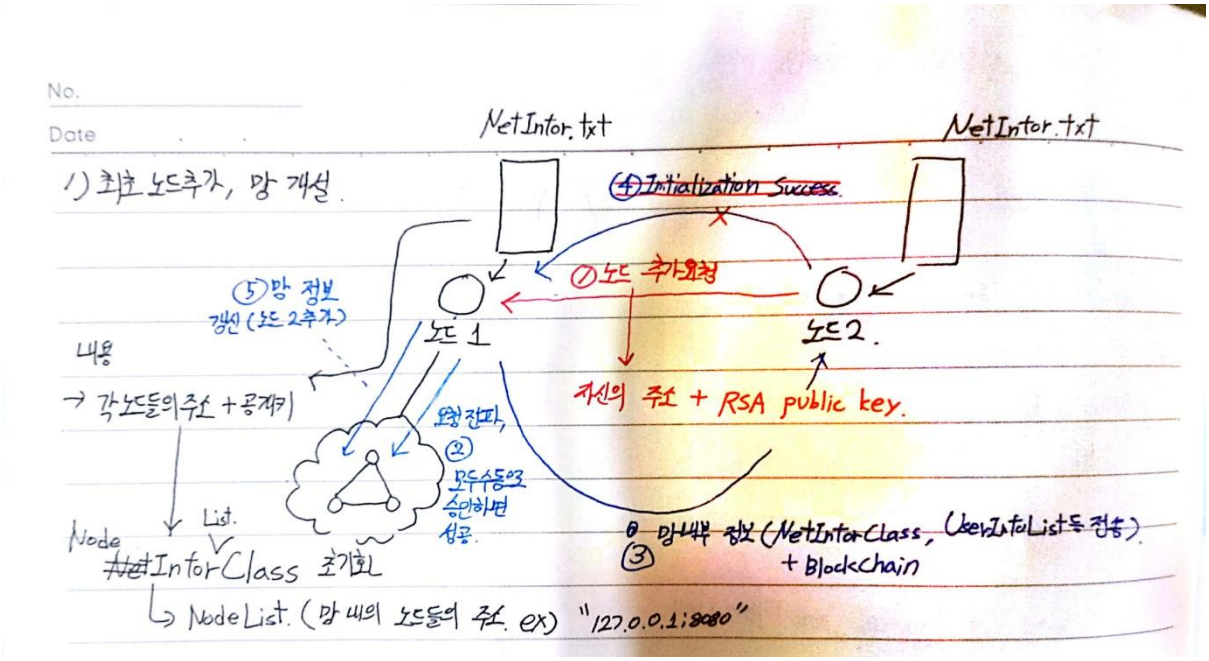
프로젝트 작동

1. 초기 노드 실행



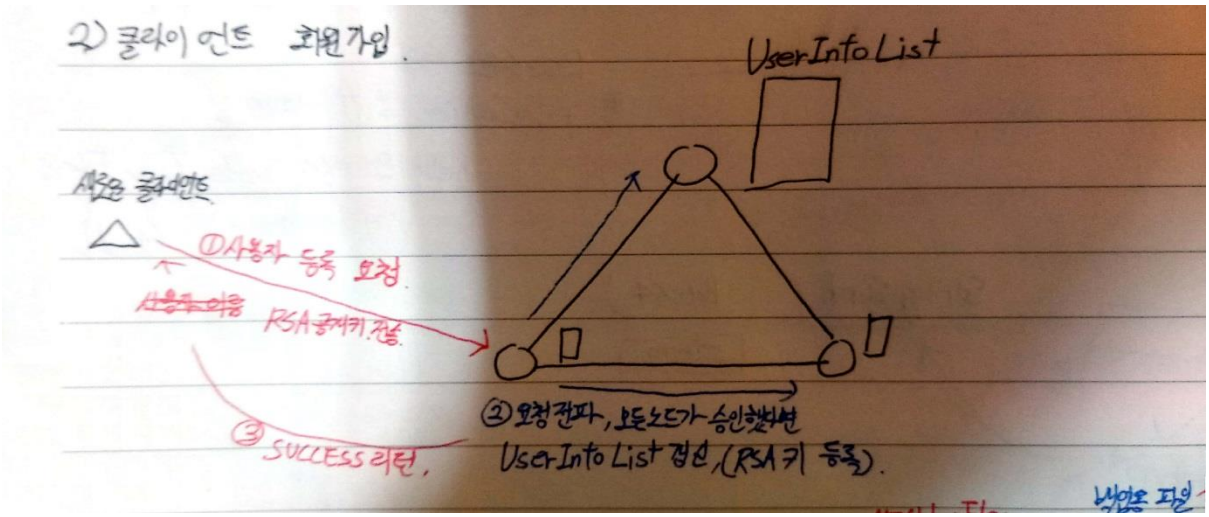
처음 노드를 실행하면 기존에 있는 망에 등록할건지 아니면 새로운 망을 생성할건지 정한다. 새로운 망을 생성하기로 선택하면 블록체인인 BlockChain을 생성하고 UserInfoList, NodeInfoList를 생성하고 초기화한다. 다음은 기존에 있는 망에 노드를 연결하는 경우이다.

1. 기존에 있는 망에 노드 추가(AddNode)



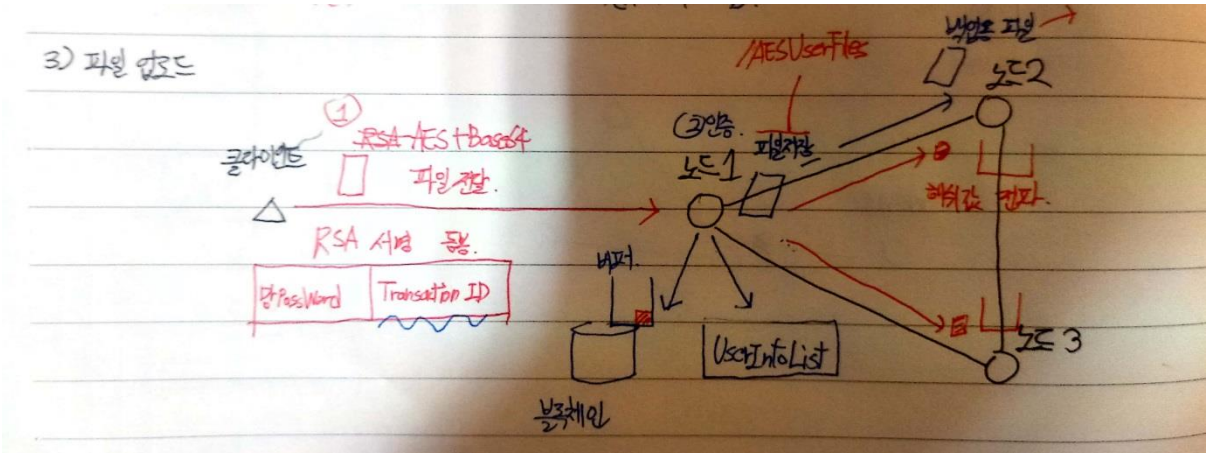
노드2를 망에 추가하는 경우이다. 노드2는 자신의 네트워크 주소와 RSA 공개키를 전달하며 등록 요청을 한다. 노드 1은 이 요청을 접수하고 망에 전파한다. 망 내의 모든 노드가 이를 승인해야 요청이 승인되고 리턴 된다. 이 시점에 망 내부정보 NodeInfoList가 갱신되고 노드2는 BlockChain, NetInforList, UserInfoList를 전송받고 망에 참여하게 된다.

2. 클라이언트 회원가입 (AddUser)



위 그림은 새로운 사용자를 망에 등록하는 과정이다. 등록받기를 원하는 클라이언트는 자신의 RSA키를 노드에 보내고 등록 요청을 한다. 이를 접수한 노드1은 이를 망에 전파하고 망 내의 모든 노드가 이를 승인해야 모든 UserInfoList가 갱신되고 사용자가 등록된다.

3. 파일 업로드 과정(UploadFile)



클라이언트가 파일을 업로드 하는 과정은 다음과 같다.

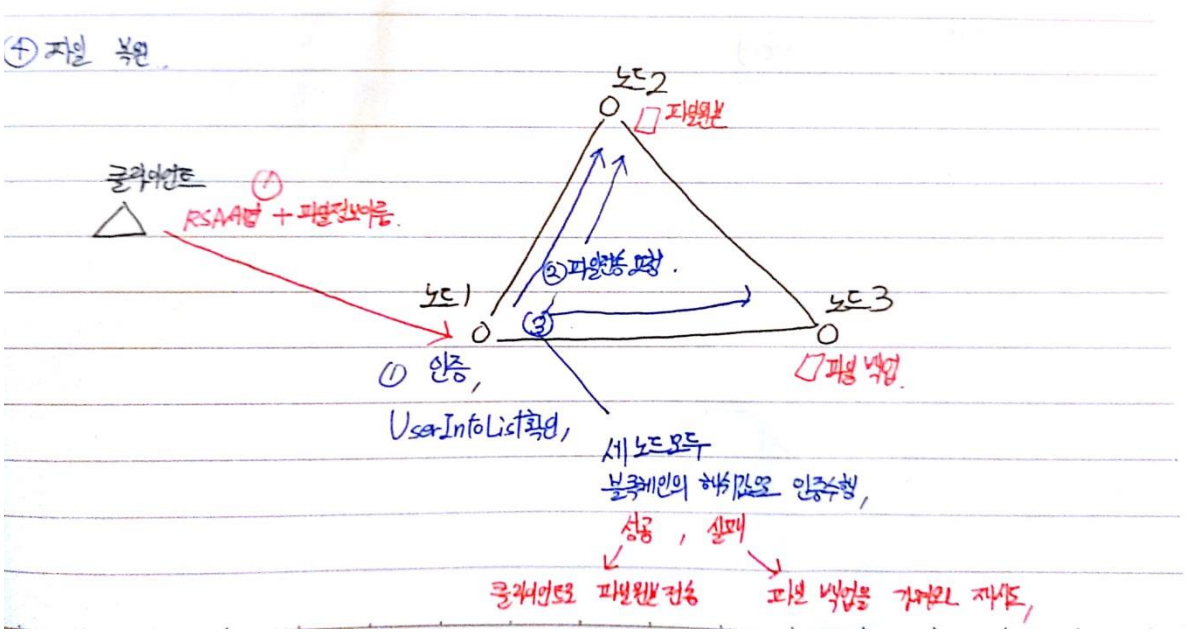
1. 클라이언트는 업로드 하려는 파일을 AES로 암호화, Base64로 인코딩, 이 후 자신의 RSA키로

작성한 서명과 함께 파일의 "이름"을 적어 노드 에게 업로드 요청을 보낸다. 이 때 서명의 내용은 "블록체인 망 비밀문자열" + "클라이언트 TransactionCount" 를 통해 생성된다. 여기서 TransactionCount 의 값은 사전에 노드 에게 물어봐서 알 수 있다.

2. 이 요청을 접수한 노드1은 먼저 보낸 공개키가 UserInfoList에 있는지 확인, 있다면 이 값으로 서명을 풀어보고 "블록체인 망 비밀 문자열" 과 "클라이언트 TransactionCount"가 유효한지 검사한다. 만약 이 과정에서 실패하면 이 요청은 무시된다. 만약 인증에 성공했다면 해당 파일을 /AESUserFiles에 "해당 사용자 키의 UserInfoList의 인덱스값" + "파일이름" 의 파일명으로 저장한다. 또한 이 파일의 복사본(백업용)을 다른 노드 중 하나에게 전달한다. 이후 해당 파일을 Base64로 디코딩, SHA-256으로 해쉬 한다. 이 값을 HashBuffer에 저장된다. 또한 이 해쉬값을 망 내의 모든 노드 에게 전파한다. 이 과정에서 모든 노드의 UserInfoList에 해당 파일의 정보(블록인덱스, 해쉬 인덱스, 원본파일 주소, 백업파일 주소)가 갱신된다. 그리고 클라이언트의 요청은 SUCCESS 값으로 리턴된다.

3. 해쉬값을 만든 노드와 해쉬값을 전파받은 노드는 모두 자신의 HashBuffer에 해당 파일의 해쉬값을 저장한다. 이 해쉬값이 5개가 달성되면 채굴을 시작한다. 또한 HashBuffer는 BlockChain의 마지막 블록의 인덱스값 + 1의 블록처럼 취급한다. (버퍼에 있는 파일 해쉬값들도 접근할수 있게 하기 위함.)

4. 파일 복원(FileRestore)



1. 클라이언트는 공개키와 서명과 가져올 "파일 이름" 을 제시하며 노드 1에게 요청을 보낸다.

2. 노드1은 서명의 인증을 수행하고 해당 사용자의 공개키를 가지고 UserInfoList를 탐색, 전달된 "파일이름" 의 정보에 접근해 해당 파일의 [블록인덱스, 해쉬인덱스, 원본파일 주소, 복구파일 주소] 4가지 정보를 획득한다. 그리고 먼저 "원본 파일 주소"를 대상으로 파일 원본을 요청한다.

파일을 전송 받았으면 블록체인에 있는 해당 파일의 해쉬값과 비교를 수행한다.

- 노드1은 원본파일의 해쉬값을 구한후 이를 노드 2와 노드3에게도 전달해 해쉬 비교를 요청한다. 결과적으로 망 내의 모든 노드가 해쉬 일치에 성공해야 SUCCESS이다.

만약 여기서 해쉬값 일치가 성공했을경우 해당 파일을 클라이언트로 전송하고 SUCCESS를 리턴한다. 만약 실패했을 경우 복구파일 주소를 대상으로 파일을 다시 얻어오고 해쉬값 비교를 다시 수행한다. 만약 여기서도 실패한다면 FAIL을 리턴한다.

이번주 목표

1. ProtocolTest 개발환경 구축 (boost, simpleweb, json 사용)

-Node 프로젝트 : 노드에서 수행해야할 프로토콜 정의

-Client 프로젝트 : 클라이언트에서 수행해야할 프로토콜 정의

2. 위에서 제시된 4가지 시나리오의 프로토콜 프로토타입 정의

- 세부 구현은 모두 제외하고 오로지 Request 와 Response 과정만 구현,

- 실행 과정 추적을 위해 중간중간 출력문 삽입하기

Simple-Web-Server 의 동기, 비동기 입출력 사용법 숙지하기

https://github.com/eidheim/Simple-Web-Server/blob/master/tests/io_test.cpp

이거 분석하기 (Simple-Web-Server Request, Response 샘플)

3. NodeInfoClass.hpp 만들기