설계과제 최종보고서

설계과제명 : 파일 기반 암호화 저장 블록체인 구현

교과목명	공개SW프로젝트_01	
담당교수	장태무	
팀 원	학 번	이 름
	2014112088	김휘건
	2014112036	안지석
	2014112060	양채훈

설계과제 요약서

과 제 요 약 서	
설계과제명	파일 기반 암호화 저장 블록체인
주요기술용어 (2~7개 단어)	블록체인, 스토리지 서비스, 암호화, P2P통신

1. 과제 목표

- 파일기반 암호화, 무결성 증명 기능을 제공하는 블록체인 구현

2. 수행 내용

- 암호화, 해시, P2P 통신, 합의 등 필요한 기능 구현

3. 수행 결과

- 다수 사용자 동시적 사용 가능한 블록체인 망 구현 성공
- 깃허브 주소 : https://github.com/CSID-DGU/2018-2-OSSP-FileEncryptionBlockChain

4. 결과 분석

- 파일 업로드, 다운로드, 암호화, 파일 무결성 검사, 등의 주요 서비스를 클라우드에서 제공,

1. 서론

1.1 과제개요

기존의 클라우드 기반의 중앙화된 스토리지 서비스에 대한 대안으로 블록체인 기반 분산 저장 스토리지를 구상해 보았다. 이 프로젝트는 중앙 서버에 파일을 저장하는 것이 아니라 블록체인 망에 참여하는 각 노드가 파일을 분산해서 저장하고 파일의 해시값만을 블록체인에 저장한다는 점이 핵심이다. 이러한 방식으로 망 전체의 스토리지 용량을 사용할 수 있게 된다. 또한 모든 블록체인 사용자들의 익명성을 보장하기 위해 파일 암호화 기능을 추가하여 오직 그 파일의 주인 만이 파일을 복호화 할 수 있도록 설계하였다.

1.2 목표 설정

블록체인 기반 스토리지 서비스를 구현하기에 앞서 다음의 다섯 가지 블록체인의 특징에 따른 목표를 설정했다.

Decentralization

특정 기업, 기관이 아니라 모두가 서로의 데이터를 암호화된 형태로 저장,
분산저장으로 용량에 이점
더 많은 백업파일 저장이 가능해짐
노드간의 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
파일 훼손 등의 에러 발생시 대처가능

Auditability

가입자의 익명 ID를 기준으로 사용자들을 관리, 저장한 파일등의 정보를 접근가능

위의 특징을 모두 만족하면서 다수의 클라이언트들에 동시적으로 서비스를 제공할 수 있는 블록체인을 구현하는 것을 목표로 설정하였다. 또한 블록체인은 추후에 얼마든지 확장 가능한 형태로 만들기 위해 최대한 동적인 형태의 소프트웨어 구조로 설계되어야 하며 입력되는 데이터의 종류는 파일로써 최대한 범용적 이어야 한다는 목표와 최대한 블록체인의 기본적 특징을 살리면서 다른 산업분야에서도 응용 및 사용가능한 프로젝트가 되도록 추가 목표를 설정하였다.

2. 배경

2.1 관련 기술의 동향

블록체인은 현재 금융분야 에서는 암호화폐 및 스마트계약 등의 기능을 활용한 증권거래, 계약서, 신용장 등의 위변조를 방지 할 수 있는 무역거래, 송금 등의 금융 업무를 수행시 규정 준수 여부를 모니터링 할 수 있는 등의 기능으로 활용되고 있고, 비 금융 분야 에서도 디지털 신원 정보를 저장하거나 전자투표, 보험사기 방지 등에 활용되고 있다. 이렇듯이 블록체인 기반 기술이 계속 발전하고 있고 이에 따라 블록체인 구현 기술 자체에 대한 수요도 계속 증가하고 있다고 볼 수 있다. 본 프로젝트는 이러한 상황에 맞춰서 위에서 언급한 서비스들의 기반이 되는 블록체인 자체를 구현하기 위한 프로젝트이다.

2.2 관련 기술의 수요 및 전망

2.1절에서 서술한 것처럼 계속해서 다양해지고 발전하는 블록체인 기반 서비스들의 수요를 맞추기 위해서는 이에 맞춰 그 기반 기능들을 제공하는 블록체인 자체에 대한 연구와 발전이 필요하다. 이와 관련된 예로 블록체인 기반 플랫폼을 제공하는 이더리움과 같은 서비스도 있다. 하지만 이더리움과 같은 코인 기반 대형 퍼블릭 블록체인과 달리 본 프로젝트 에서는 어느 기관이나 단체에서 인트라넷 등과 같은 내부망에서 동작할 수 있는 자체적인 프라이빗 블록체인을 목표로 하고 다양한 데이터를 입력으로 취할 수 있는 범용적인 구조를 목표로 한다.

3. 제한요소

주요한 제한 요소는 구현한 블록체인의 테스트가 힘들다는 것이다. 제대로 블록체인이 동작하는가를 확인하려면 어느 정도 규모가 있는 P2P 네트워크를 구성하고 다수의 클라이언트를 가지고 시뮬레이션을 수행해야 한다. 이런 제한 사항을 감안하고 구글 클라우드 서비스에서 최대 세 개의 가상 인스턴스에 노드 프로그램을 실행시킨 뒤 소규모 프라이빗 블록체인을 구성하고 작업하기로 하였다. 다른 제한 요소는 동시성 문제 검증의 애매함이다. 블록체인의 각 노드는 다수의 요청과 응답을 동시적으로 수행할 수 있는 형태로 작동해야 한다. 이 부분을 안정화하기 위해서 개발기간 이외로 다양한 상황에서의 테스트 기간이 필요하지만 전체적인 프로젝트 기간이 부족하다. 따라서 블록체인이 소수의 클라이언트의 요청에 따라 적절한 응답을 오류 없이 동작하는 것을 목표로 설정하였다.

3.1 개발환경

모든 조원들이 익숙하게 사용할 수 있는 운영체제, 언어와 IDE를 고려해서 윈도우 환경, C++, VisualStudio2017을 사용하기로 하였다.

3.2 동작환경

프로젝트는 노드 프로그램과 클라이언트 프로그램으로 나뉘고 전자는 리눅스 환경에서, 후자는 윈도우 환경에서 컴파일되고 실행된다. 노드 프로그램은 서로 다른 세 개의 구글 클라우드 플랫폼 위에서 동작한다. 만약 구글 클라우드에서 작동시키는 것이 여의치 않으면 단일 윈도우 환경에서 로컬로 실행한다. 두 프로그램은 리눅스, 윈도우 환경에서 모두 컴파일 될 수 있도록 C++표준 코드로 작성되어야 한다.

4. 설계

4.1 설계안

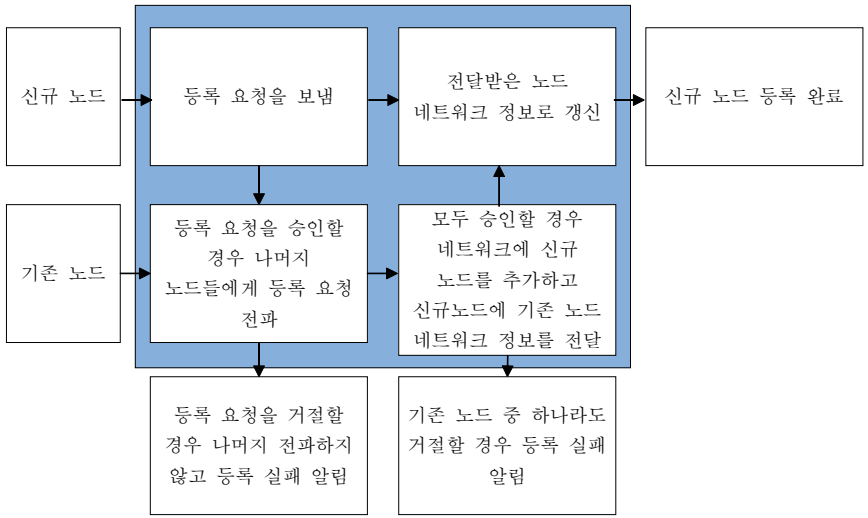
블록체인에서 이루어 져야할 시나리오를 크게 세 가지로 정리하고 이에 따른 세부 설계를 구상해 보았다. 다음 절은 “블록체인 네트워크에 새로운 노드가 접속하는 경우”, “클라이언트가 파일을 업로드 하는 경우”, “클라이언트가 파일을 다운로드 받는 경우” 세 가지 시나리오 대한 설계이다.

4.1.1 블록체인 네트워크에 새로운 노드가 접속하는 경우

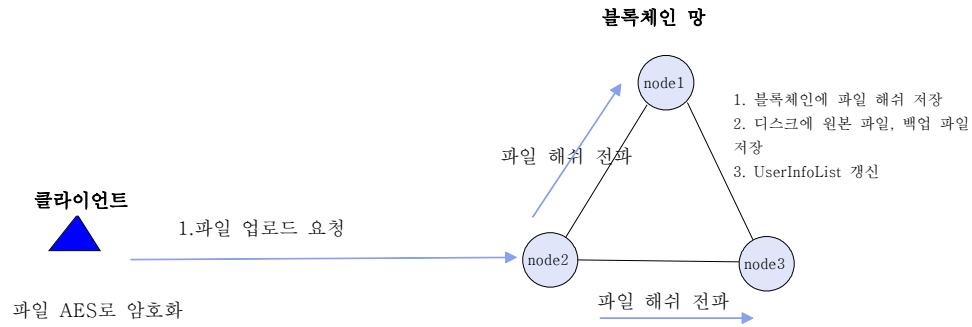
노드 등록 성공
신규 등록하려는 노드가 기존 노드들 중 하나의 노드에게 등록요청을 보냄
→ 요청받은 노드가 승인하면 기존 노드에 연결되어 있는 나머지 노드들에게 요청 전파
→ 모든 노드가 승인할 경우 기존 노드 네트워크에 신규 노드를 추가
→ 신규 노드에 기존 노드 네트워크 정보 전달
→ 신규 노드 네트워크 정보 갱신
→ 등록성공

노드 등록 실패 1
신규 등록하려는 노드가 기존 노드들 중 하나의 노드에 등록요청을 보냄
→ 요청받은 노드가 거절
→ 등록 실패 알림

노드 등록 실패 2
신규 등록하려는 노드가 기존 노드들 중 하나의 노드에게 등록요청을 보냄
→ 요청받은 노드가 승인하면 기존 노드에 연결되어 있는 나머지 노드들에게 요청 전파
→ 기존 노드들 중 하나라도 거절
→ 등록 실패 알림

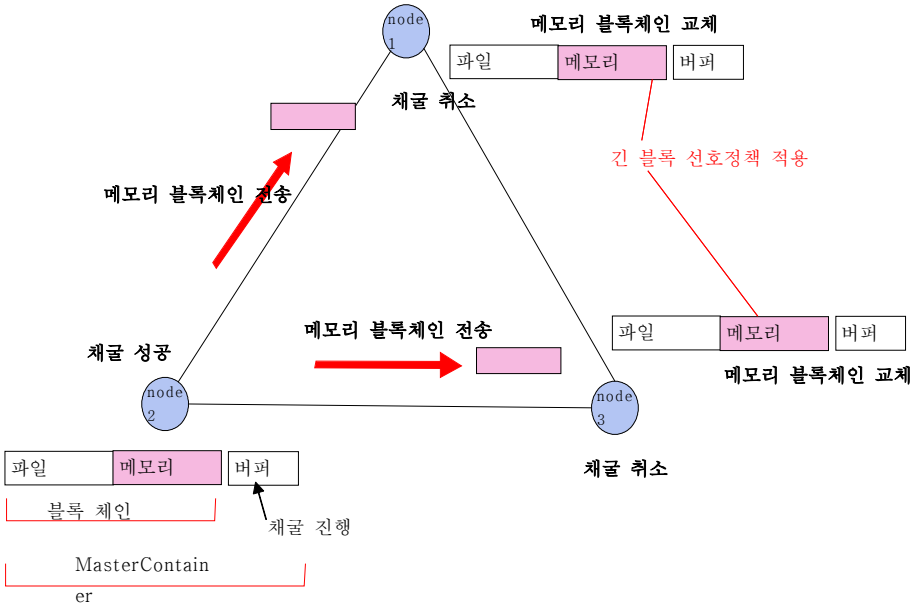


4.1.2 클라이언트가 파일을 업로드 하는 경우



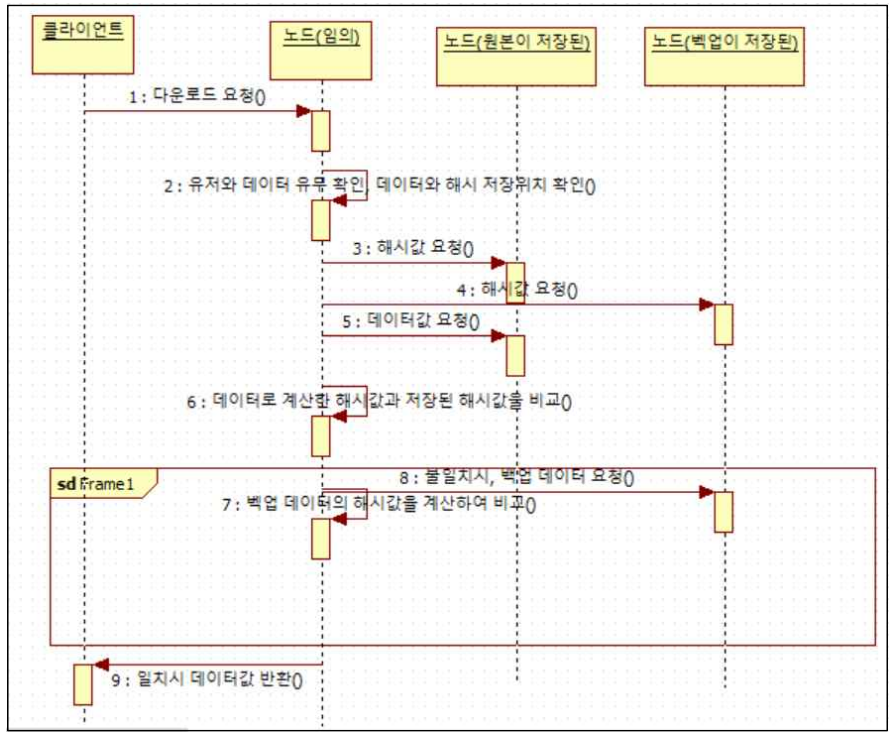
클라이언트는 파일을 AES로 암호화 한후 블록체인에 업로드 한다. 이 요청을 받은 노드는 기본적인 인증 과정을 거친후 파일 해쉬를 계산, 블록체인 네트워크에 전파한다.

4.1.3 채굴 및 블록 전파



파일의 해시값들은 MasterContainer라는 자료구조에 입력된다. 이 자료구조는 블록체인과, 채굴 기능을 담당하고 있다. 블록체인의 경우 일정량은 메모리에 저장하고 일정량을 초과하면 파일에 저장한다. 입력된 파일 해시값들이 버퍼에 쌓이면 채굴을 시작하고 블록을 만들어 전파한다. 이때 긴 블록 선호 정책이 적용되며 자신이 현제 가지고 있는 블록체인 보다 더 긴 길이의 블록체인이 전파되었을 때만 블록체인 교체를 수행한다.

4.1.4 파일 다운로드



1.클라이언트가 임의의 노드에게 사용자 명과 다운로드할 데이터 명을 전송하며 파일 다운로드를 요청한다.
2.요청을 받은 노드는 자신의 userinfolist에서 유저와 데이터의 유무를 확인하고 해당 데이터를 저장한 노드와 해시의 위치를 확인한다. (만약 없을 경우 에러처리)
3,4 망에 참여한 다른 노드들에게 해시값을 요청한다(RequestHash)
5. 3,4와 병행하여 원본 데이터가 저장된 노드에게 데이터를 요청한다.
6. 3,4,5가 끝나면 다운받은 데이터의 해시를 계산하여 블록의 해시값과 비교한다.
7. 불일치 시, 백업 데이터를 다운받는다.
8. 백업 데이터의 해시값을 계산하여 비교한다.
9. 5나 9에서 일치하는 경우 정상이므로 데이터 값을 반환한다.
AES로 암호화되어 있기 때문에, 클라이언트에 보관중인 키를 사용하여 복호화 한다.

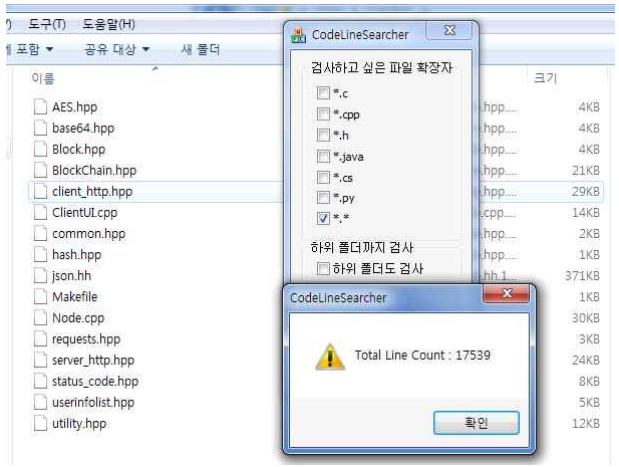
4.2 대안 분석

	대안	프로젝트 계획으로 채택
클라이언트 UI	텍스트 기반의 UI로 할 경우 구현이 간단하다는 장점이 존재하지만, 사용자 입장에서 보았을 때 기능을 한눈에 알기 어렵고 익숙하지 않은 사용자는 사용하기 힘들다는 단점이 있다.	그래픽 기반의 UI로 할 경우 구현을 위해 더 많은 노력이 필요하며, UI설계에 다양한 유형이 있기 때문에 선택에 어려움도 있지만, 잘 구현된 UI의 경우 사용자에게 친근감을 주며 익숙하지 않은 사용자도 손쉽게 사용할 수 있다는 장점이 있다.
파일 암호화	RSA로 암호화 하는 경우 전자서명 등의 기능을 사용할 수 있다는 장점이 있지만, 공개키와 비밀키를 모두 관리해야하는 번거로움이 있다.	AES로 암호화 하는 경우 전자서명 등의 기능을 사용할 수 없다는 단점이 있지만, 하나의 키로 암호화와 복호화가 모두 가능하기 때문에 키관리가 간편하다.
프로토콜	통신을 위한 프로토콜을 별도로 자체계발 할 경우 프로젝트에 최적화된 프로토콜을 만들어 성능을 향상시킬 수 있지만, 구현을 위해 많은 비용이 소모되고 동시에 기존의 통신 프로토콜과 호환이 어렵다는 단점이 있다.	통신을 위해 기존의 http 클래스 이용할 경우 프로젝트에 최적화되지 않은 범용적인 프로토콜이기에 성능의 감소가 있다는 단점이 있지만, 구현을 위한 별도의 비용이 불필요하고 기존의 통신 프로토콜과 호환이 가능하다는 장점이 있다.
RSA 서명을 통한 보안기능	RSA 서명을 통한 보안 기능을 추가한다면, 블록의 소유권에 대한 기능을 기반으로 한 보안기능이 사용 가능하지만, 구현을 위해 RSA 키를 사용해야 하며 사용된 키의 replay attack을 막기 위한 별도의 카운터 값을 유지해야 하기 때문에 구현이 복잡해지는 단점이 있다.	RSA 서명을 통한 보안 기능을 생략한다면, 블록의 소유권에 대한 기능을 기반으로 한 보안기능이 사용 불가능 하지만, 기존의 AES키를 이용하여 데이터를 암호화 하고 있고, 블록의 소유권에 대한 정보를 대신하여 자신의 저장권한에 대한 값을 별도로 유지하고 있기 때문에 기존의 기능에는 큰 문제가 없다.
블록체인에 저장할 요소(용량,속도, 무결성등을 고려)	파일 전체를 블록에 저장할 경우, 블록의 용량이 매우 커지며, 통신에 필요한 시간이 길어지는 단점이 있지만 무결성의 경우 긴 블록 선호 정책에 의해, 위에 어느 정도의 블록이 쌓인 경우 변경되지 않는다. 하지만 위에서 언급된 블록의 용량이 매우 커지는 문제로 인해 시간이 지날수록 블록의 유지에 필요한 데이터의 크기가 커지기 때문에 블록의 유지가 어렵다.	파일의 해시값만을 블록에 저장할 경우, 블록의 크기가 적어져서 시간이 지나도 블록의 유지가 용이하며, 전송되는 데이터가 적기 때문에 통신 속도도 상대적으로 빨라지며, 무결성에 대한 부분은 파일의 해시값을 블록에 저장하기 때문에 무결성에 대한 기능은 파일 전체를 저장하는 경우와 동일하다.

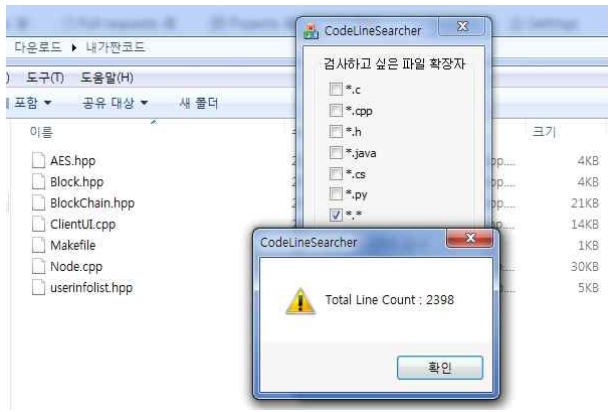
5. 구현

깃허브 저장소 : https://github.com/CSID-DGU/2018-2-OSSP-FileEncryptionBlockChain

프로젝트 전체 소스코드 라인 수 : 17539줄



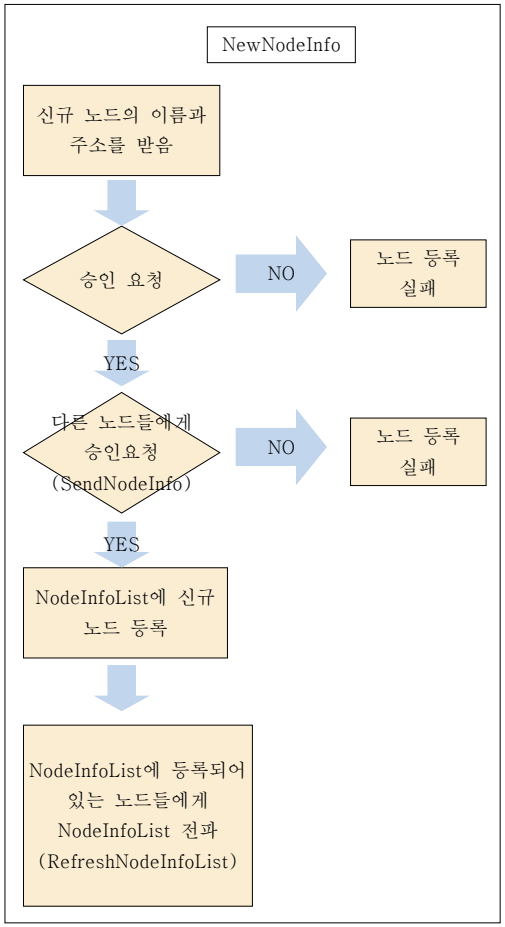
직접 작성한 소스코드 라인 수 : 2398줄



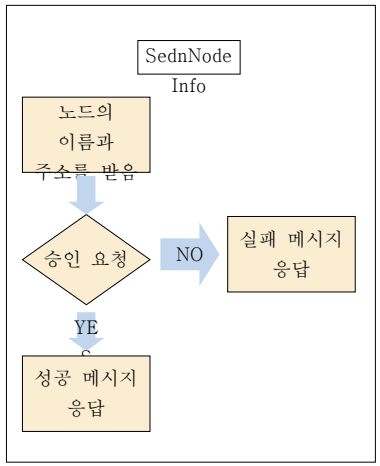
5.1 구현 방법

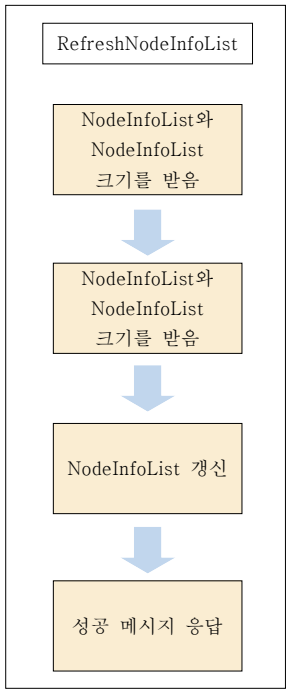
블록체인에서 이루어지는 시나리오는 크게 3가지로 나눌 수 있다. 다음 절에서 순서대로 “블록체인 네트워크에 새로운 노드가 접속하는 경우”, “클라이언트가 파일을 업로드 하는 경우”, “클라이언트가 파일을 다운로드 받는 경우” 세 가지 시나리오에 대해 설명한다.

5.1.1 블록체인 네트워크에 새로운 노드가 접속하는 경우

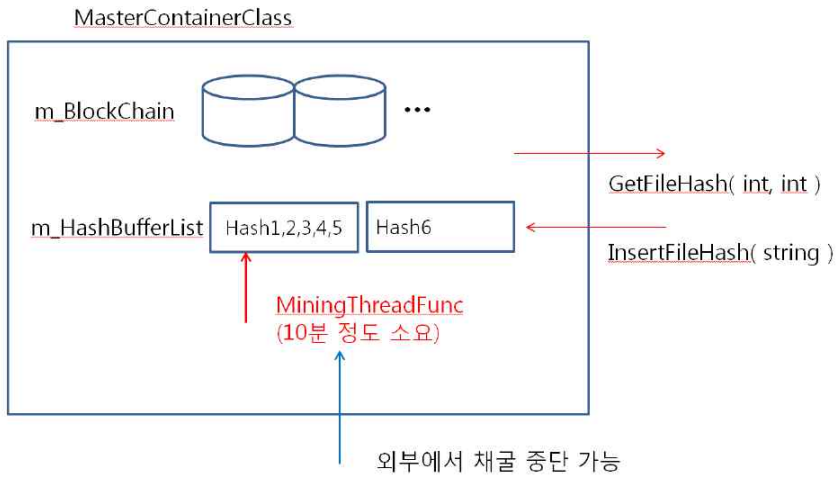


위 그림은 블록체인에 참여하고자 하는 노드에서 전송한 NewNodeInfo 의 처리 및 합의 과정이다. 위 프로토콜에서 등장하는 SendNodeInfo, RefreshNodeInfoList의 처리는 다음과 같다.





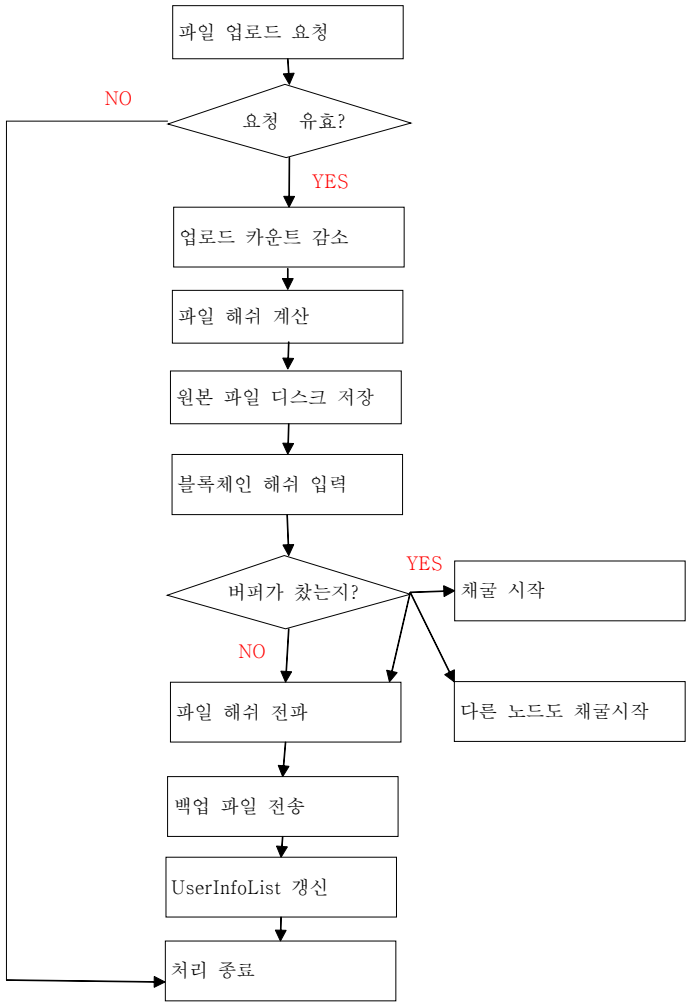
5.1.2 블록체인 파일 업로드



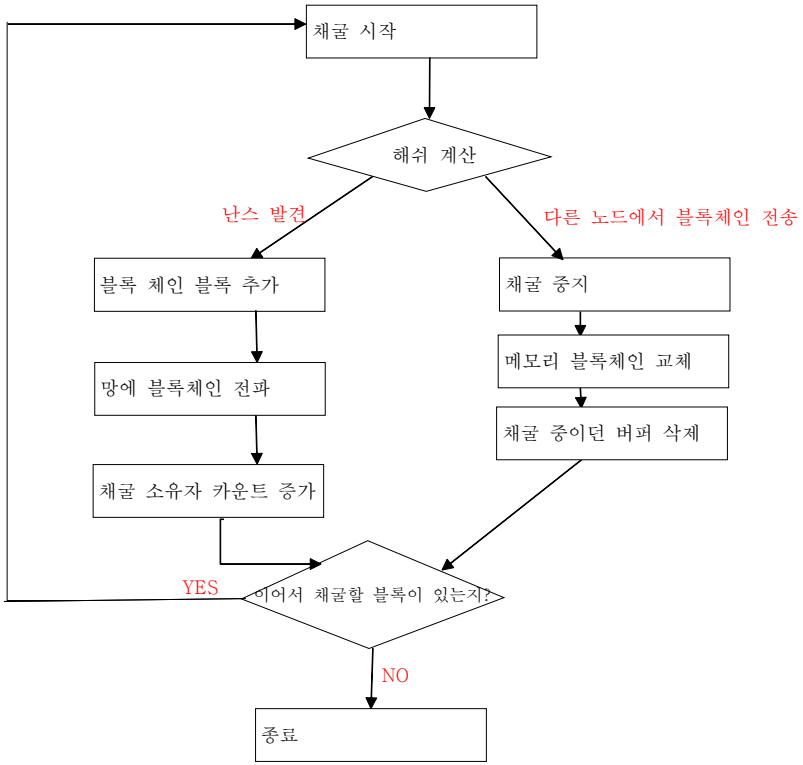
위 그림은 파일 해쉬 블록체인 및 채굴 기능을 담당하고 있는 MasterContainer 클래스 이다. GetFileHast 함수는 파일 해쉬를 가져올 때 사용하고 InserFileHash 함수는 파일 해쉬를 입력할 때 사용한다 m_BlockChain 멤버는 파일 해쉬를 저장하는 블록체인 이고 m_HashBufferList는 채굴 해야할 파일 해쉬들이다. 채굴기능을 담당하고 있는 함수는 MiningThreadFunc 이다.

다음은 파일 업로드 요청, 채굴 시작 두 시나리오에 따른 플로우 차트이다.

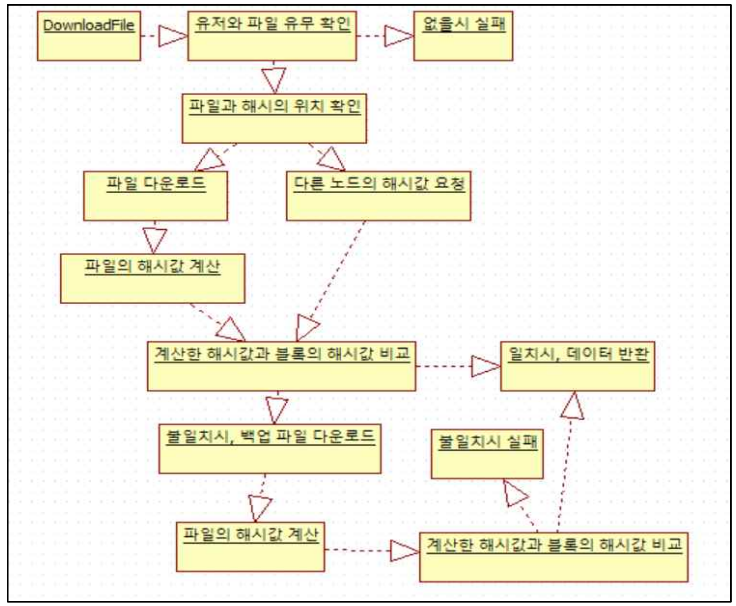
<파일 업로드 요청>



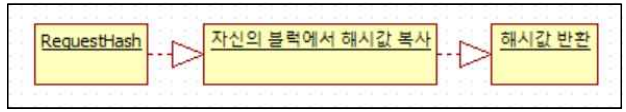
<채굴 시작>



5.1.3 블록체인 파일 다운로드



위 그림은 사용자가 DownloadFile 요청을 노드로 전송했을 때의 처리이다. 위 과정에서 “다른 노드의 해시값 요청” 과 "파일 다운로드“ 에 따른 처리는 다음과 같다.



위 그림은 DownloadFile중 ‘다른 노드의 해시값 요청’에서 실행되며 블록 인덱스, 해시 인덱스를 전송받아 자신의 블록에서 해당 하는 해시값을 반환한다.

위 그림은 DownloadFile중 ‘파일 다운로드’, ‘백업 파일 다운로드’에서 실행되며 파일명을 전송받아 자신의 메타데이터에서 해당하는 데이터를 스트링으로 변화하여 반환한다.

5.2 구현 도구

사용언어 : C++

개발툴 : VisualStudio2017

사용 라이브러리 : boost, OpenSSL, Crypto++, NaNa

사용 오픈소스 : nlohmann/json(json parser), eidheim/Simple-Web-Server (HTTP library),

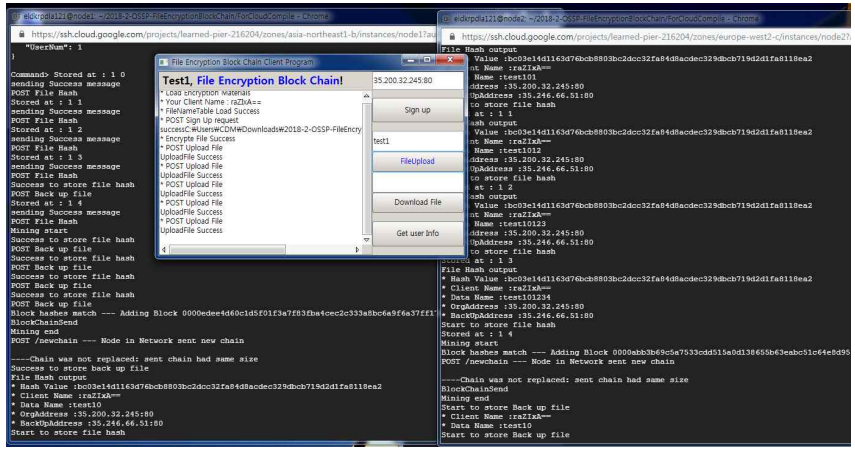
6.1. 설명

다음은 프로그램의 다양한 상황에서의 실행 결과이다.

6.1.1 블록 체인 내용 출력

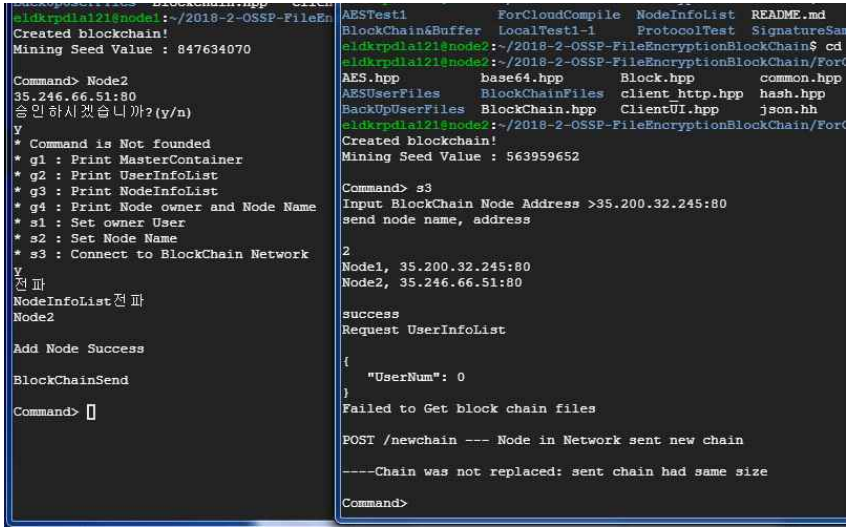
위 사진은 메모리에 저장중인 블록체인의 내용을 출력한 것이다. 상단에 표시된 "BufferList"는 채굴 대기중인 파일 해쉬값들이다 현재는 하나의 파일 해쉬가 들어가 있으며 5개 단위로 채굴을 수행하게 된다. 계속해서 "blockchain" 는 이미 채굴이 완료되어 블록체인에 들어가있는 파일 해쉬값들을 출력하고 있다. 위 정보는 메모리에 저장되어 있는 블록체인 만을 출력한것이며, 메모리에 저장할 수 있는 블록의 수는 동적으로 설정 가능하다. 해쉬로는 SHA256을 사용한다.

6.1.2 파일 업로드 +채굴 + 블록 전파 + 긴 블록체인 선호 정책



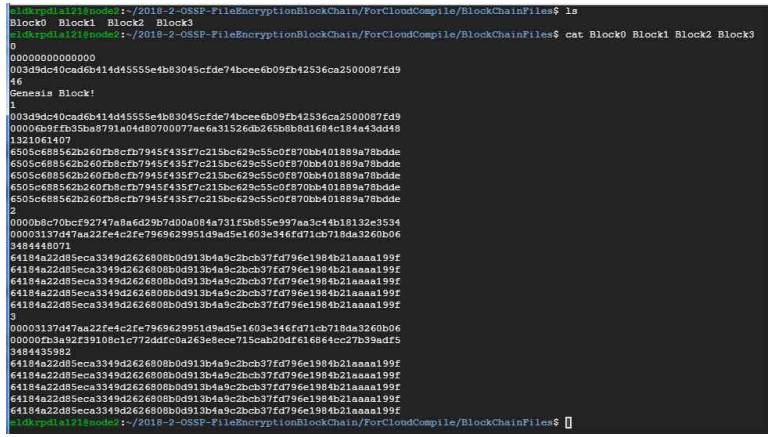
위 사진은 블록체인 망의 노드를 대상으로 다섯 번의 파일 업로드 요청을 보낸 것이다. 파일 해쉬가 다섯 개가 입력됨에 따라 망에 참여중인 모든 노드는 해쉬를 계산하게 되고 먼저 난스값을 발견한 노드는 블록을 만들어 전파하게 된다. 위의 사진의 경우에는 동시에 난스값이 발견된 경우이고 이 경우 두 노드는 독자적으로 서로 다른 블록체인을 구성하게 된다. 또한 전파한 블록체인은 긴 블록 선호 정책에 따라 반영되지 않고 폐기된다. (이미 두 노드가 블록을 블록체인에 추가했으므로 전파한 블록체인과 길이 차이는 없다. 따라서 더 긴 블록체인만을 반영하는 긴 블록선호 정책에 위배되므로 폐기한다.) 파일 업로드 횟수는 사용자 별로 제한되어 있으며 채굴 보상을 통해 증가시킬 수 있다.

6.1.3 노드 접속



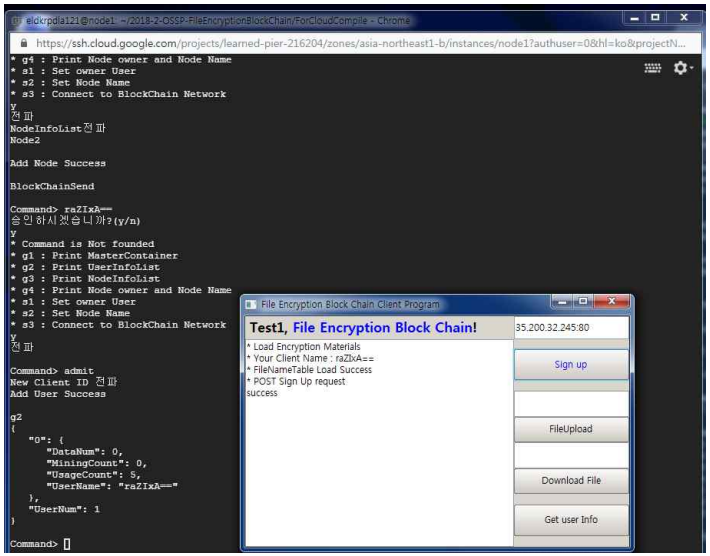
위 사진은 좌측의 노드1번에 우측의 노드2번이 접속하는 상황으로 노드1이 승인한다면 노드2는 블록체인과 메타데이터를 다운로드받고 블록체인 망에 참여하게 된다.

6.1.4 파일에 저장된 블록체인



위 사진은 메모리가 아닌 파일에 저장되어 있는 블록체인의 블록들을 출력한 것이다. 메모리에 채굴된 블록들이 일정량을 넘어가면 파일로 저장하게된다. 포맷은 라인별로 다음과 같다. 1.블록 인덱스, 2.이전 블록 해쉬, 3.현재 블록 해쉬, 4.난스값, 5.다섯개의 파일해쉬값

6.1.5 사용자 인증 절차

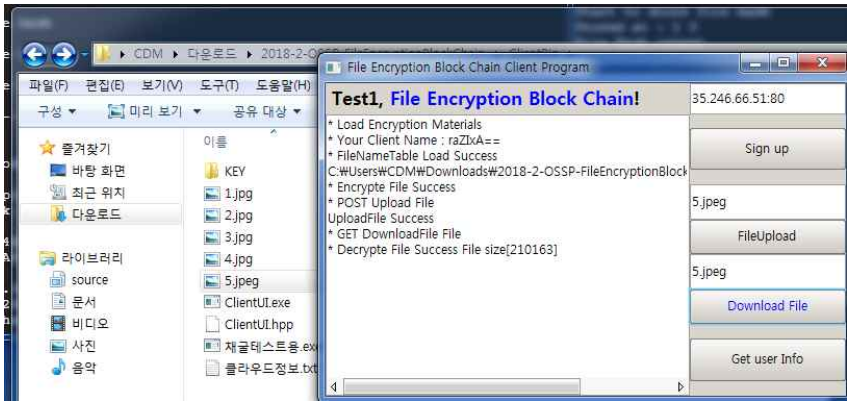


블록체인의 파일 업로드, 다운로드 기능을 사용하려면 클라이언트 프로그램을 사용해 인증 절차를 거쳐야 한다. 인증 절차는 클라이언트에서 임의의 노드로 인증 요청을 보내면 노드는 이를 전파해 망의 모든 노드가 이를 승인해야 인증이 완료되는 형태이다. 위의 사진의 경우 랜덤으로 생성된 Client ID인 "raZIxA==" 라는 사용자가 35.200.32.245:80 이라는 노드를 대상으로 인증 요청을 보내고 노드는 이를 승인하여 사진의 하단에 raZIxA== 라는 사용자가 블록체인 망에 등록되었음을 확인할 수 있다.

6.1.6 암호화 되어 저장되어있는 파일

위의 사진은 블록체인 망의 임의의 노드에 저장되었는 어떤 사용자의 어떤 파일을 출력한 것이다. 파일의 내용은 클라이언트 프로그램에서 AES로 암호화 된후 Base64인코딩 과정을 거쳐 임의의 노드의 파일에 저장된다.

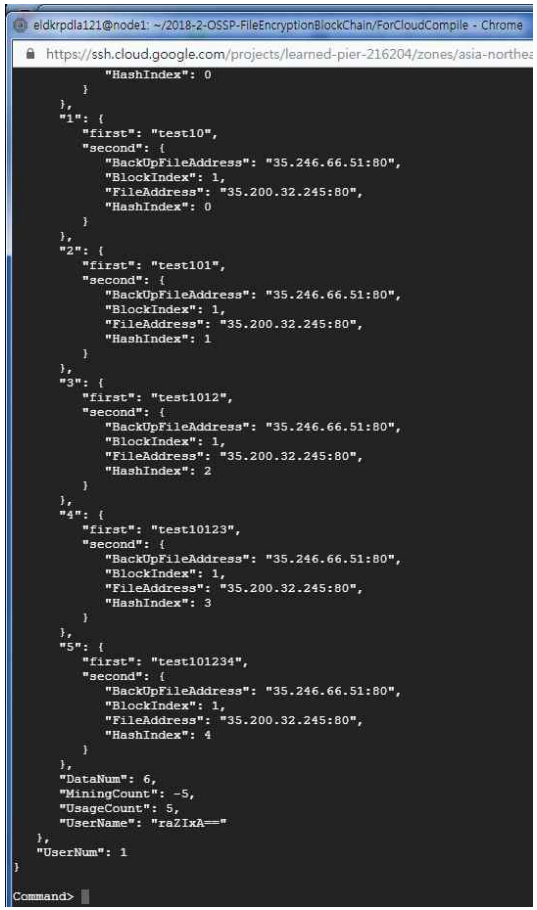
6.1.7 파일 다운로드



위 사진은 클라이언 프로그램에서 5.jpeg라는 심볼로 저장된 파일을 다운로드 요청하는 모습이다. 블록체인 네트워크는 이에 해당하는 파일을 블록체인에 저장된 파일해쉬를 통해 무결성 검사를 거친후 이를 클라이언트 프로그램으로 전송해준다. 이 과정에서 해쉬를 통한 무결성 검사는 모든 노드가 수행한다. 만약 해쉬가 불일치 할 경우 원본 파일은 폐기하고 백업파일을 가지고 다시 인증 과정을 수행한다. 모든 백업파일이 전부 무결성 검사에 실패할 경우 해당 파일 다운로드 요청은 실패한다. 위 사진에서는 파일을 다운로드 받은후 Base64로 디코딩, AES로 복호화를 수행하고 클라이언트 프로그램에 저장되어있는 파일 원본의 경로, 파일이름을 가지고 파일을 저장한다.

6.1.8 사용자 정보 조회

1.노드



위 사진은 노드 프로그램에서 UserInfoList 라는 자료구조를 출력한 것이다. 이 자료구조는 사용자와 사용자가 저장한 파일에 대한 정보를 모두 기록 하고있는 메타데이터이다.

2. 클라이언트

클라이언트 프로그램에서는 "Get user Info" 라는 버튼을 통해 사용자 정보를 조회할 수 있다.

6.2. 시험 및 평가

평가항목	평가방법	평가
채굴 작동, 전파 여부	다수의 파일 업로드 요청을 전송한다.	채굴성공, 블록 전파성공
긴 블록체인 선호 정책 적용 여부	두 노드가 동시에 채굴을 완료할 경우의 처리를 확인한다	긴 블록선호 정책 적용 성공
파일 업로드, 다운로드 정상 작동 여부	클라이언트에서 파일을 업로드 하고 다운로드 한다	원본 파일 손상없이 그대로 저장 성공
무결성 검사 정상 작동 여부	다운로드 과정에서 파일 해쉬 일치 여부를 검사한다	모든 노드가 정상적으로 합의 과정에 참여 성공
파일이 분산 저장 여부	파일이 각 노드 별로 골고루 저장되는지 검사	노드끼리 자체적인 파일 분산 기능 필요
파일 암호화	AES + Base64 형태로 저장되는지 여부	해당 포맷으로 정상 저장 확인 완료
사용자, 노드 인증 기능	인증, 합의 과정이 합리적인지 논의	단순한 키 입력으로 인증을 처리하므로 허술함
동시적인 P2P 통신여부	모든 시나리오에서의 쓰레드가 동시적으로 정상 작동하는지 검사	채굴 쓰레드에서의 동기화의 부재, 테스트의 부족, 채굴중 노드 접속등 다양한 문제 발견

6.3 기대성과

프로젝트는 향후 확장을 위해서 최대한 확장가능하고 범용적인 형태로 설계하였다. 블록체인에 저장할 수 있는 파일에는 어떤 내용도 담길 수 있으며 이를 활용하면 블록체인을 필요로 하는 다양한 분야에 적용할 수 있다.

7. 작업진행 방법

7.1 설계 일정 및 역할 분담

	작업이름	10. 2018			11. 2018				12. 2018				담당자
		2주차	3주차	4주차	1주차	2주차	3주차	4주차	1주차	2주차	3주차	4주차	
1	암호화폐 개발 진행,취소												전원 참여
2	새로운 프로젝트 설계												전원 참여
3	기본설계 완성, 개발시작												전원 참여
4	전원 개발환경 설정완료												전원 참여
5	세부 프로젝트 설계완성												전원 참여
7	3번 프로토콜 완성												김휘건
8	4번 프로토콜완성												안지석
9	1,2,번 프로토콜 완성												양채훈
10	3번 기능 완성												김휘건
11	4번 기능 완성												안지석
12	세부 기능 완성												전원 참여
13	테스트 및 문서화												전원 참여
15	프로젝트 종료												

(* 1번 프로토콜은 노드 등록 시나리오, 2번 프로토콜은 사용자 등록 시나리오, 3번 프로토콜은 파일 업로드 시나리오, 4번 프로토콜은 파일 다운로드 시나리오 이다.)

8. 결론

8.1 결론

설계단계에서 설정한 주요목표인 블록체인 상의 파일 분산저장, 암호화, 무결성 증명, 익명성 제공 등의 기본 목표는 전부 달성하였다. 하지만 확장성 문제에 대해서는 아직 해결해야할 문제가 남아 있다. 이 문제는 8.2 절에서 다룬다.

8.2 향후 계획

다음은 프로젝트 개선을 위해서 앞으로 진행할 수 있는 일곱 가지 개선점을 설명한 것이다.

8.2.1 네트워크 보안 문제 해결 필요,

http 통신 라이브러리를 사용했으므로 블록체인 내의 모든 트래픽은 네트워크에 그대로 노출된다. 이로 인해 클라이언트, 노드로 가장한 공격이 수행될 수 있으며 리플레이 어택 등의 공격에도 무방비하다.

8.2.2 P2P 통신 프로토콜 개선 필요

현재 블록체인 네트워크 상의 모든 통신은 동기적으로 이루어지고 있다. 이 부분은 비동기 통신을 사용한다면 성능을 더욱 개선할 수 있다.

8.2.3 쓰레드 동기화 버그 수정 필요

블록체인 내의 모든 멀티쓰레드 관련 코드에는 동기화 코드가 적용되어 있지 않다. 이 부분은 블록체인이 확장되었을 때 버그를 일으킬 가능성이 다분하다.

8.2.4 합의 과정을 다수결로 수정

현제 블록체인 내의 모든 합의 과정은 모든 노드가 만장일치로 승인해야 작동하도록 구현되어 있다. 이 부분은 블록체인이 확장되었을 때 다양한 상황에 유연하게 작동하지 않을 수 있다.

8.2.5 특정 노드가 비정상 종료 했을 경우의 에러 처리 필요

현재 블록체인의 구현은 특정 노드가 어떤 이유로든 종료 되었을때의 에러처리가 되어있지 않다. 만약 블록체인이 작동중 하나의 노드라도 종료되면 블록체인은 정상작동하지 않는다.

8.2.6 디폴트 채굴 기능 추가

현재 채굴 기능은 파일 해쉬가 충분히 블록체인 네트워크에 전파 되었을때 작동한다. 만약 블록체인에 충분한 파일 업로드 요청이 올라오지 않는다면 각 노드는 아무 작업도 하지않는 유휴 시간이 증가하게 된다. 이러한 상황을 방지하고 쉬는 시간 없이 채굴함수를 실행하기 위해 실제로 블록체인에는 반영되지 않는 디폴트 블록 채굴 기능을 추가해야한다.

8.2.7 백업 파일의 메타데이터 자료구조 수정

블록체인 내의 특정 파일에 대해서 충분한 양의 백업 파일을 망에 분배하는 것은 블록체인 내의 파일의 안전을 보장하기 위한 핵심적인 기능이다. 현제 프로젝트의 구현에서는 백업파일을 하나만 생성하도록 작성되어 있다. 이 부분은 망을 구성하는 노드에 따라 동적으로 변경되도록 관련 자료구조와 코드를 수정해야 한다.

9. 참고문헌

[1] An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends
Zibin Zheng1, Shaoan Xie1, Hongning Dai2, Xiangping Chen4, and Huaimin Wang3, 2017.
[2] 4차 산업혁명을 이끌 또 하나의 기술 블록체인 펼쳐보기, 김석원