

블록체인 기반 출입통제 시스템

팀 명 : Well-made

지도 교수 : 양환석 교수님

팀 장 : 조영선

팀 원 : 김민정

김예진

이정한

조수홍

2019. 10.

중부대학교 정보보호학과

목 차

1. 서론

1.1 연구 목적 및 주제 선정	2
-------------------------	---

2. 관련 연구

2.1 블록체인 종류	2
2.2 P2P 네트워크	3
2.3 PBFT 합의 알고리즘	3
2.4 Hash	4
2.5 인증서	5
2.6 얼굴인식	5

3. 본론

3.1 시스템 구성	6
3.2 프로그램 구성	7
3.2.1 인증체계	7
3.2.2 블록체인 생성과정	8
3.2.3 출입 통제	10

4. 결론	10
-------------	----

5. 별첨

5.1 참고자료	10
5.1.1 소스코드	11
5.1.2 발표자료	49

1. 서론

1.1 연구 배경 및 주제 선정

제 4차 산업혁명 시대는 정보통신기술(ICT)의 융합으로 이루어낸 새로운 기술 혁신을 의미한다. 4차 산업혁명과 함께 블록체인 기술이 미래 유망 기술로 대두되고 있다.

네트워크에 참여하는 개인과 개인의 거래(P2P)의 데이터가 기록되는 장부를 '블록'이라 하며, 이런 블록들은 형성된 후 시간의 흐름에 따라 순차적으로 연결된 '체인(사슬)'의 구조를 가지게 된다. 블록체인의 가장 큰 특징은 분산저장을 한다는 점이다. 블록체인은 네트워크 내 모든 사용자가 거래내역을 저장하고 증명한다. 기존거래 방식인 은행이 중간 역할을 하는 것이 아닌 여러 명이 나눠서 저장하고 있어 거래 내역을 확인할 때는 모든 사용자가 보유한 장부를 대조하고 확인해야 하기 때문에 거래내역을 위·변조하기 어렵다. 최근 몇몇 은행 전산망 해킹 사건이 일어났다는 점과 비교하면 보안적인 측면에 강하다.

또한, 최근 얼굴인식 기술은 공항 출입국 관리, 출퇴근 관리, 아파트 출입 관리, 범죄 수사 등 다양한 분야에서 활용된다. 얼굴인식 과정은 카메라를 사용하여 사람의 얼굴을 캡처[Capturing]한 후에 눈과 눈썹, 코와 입, 턱 등의 얼굴 특징이 변하는 각 부위 60여 곳을 분석하여 특징[Feature]되는 데이터를 추출한다. 추출된 데이터와 데이터베이스에 저장된 얼굴의 특징 데이터와 비교하여 일치 여부를 통해 얼굴을 인식한다. 이러한 얼굴인식은 개인 인증을 위해서도 활용이 된다. 얼굴인식을 통해 사용자를 판별하여 인증에 성공한 사용자만이 출입이 통제된 구역에 출입할 수 있으며 출입을 시도할 때 발생한 기록을 블록체인에 저장하여 데이터 위변조에 대한 위험을 줄이는 출입통제 시스템을 구현하였다.

2. 관련 연구

2.1 블록체인 종류

블록체인은 누구나 할 수 있는 개방형인 퍼블릭(public) 블록체인과 제한된 사용자만 참여할 수 있는 폐쇄형인 프라이빗(private) 블록체인으로 나눌 수 있으며 설명은 아래와 같다.

2.1.1 퍼블릭(public) 블록체인

퍼블릭(public) 블록체인은 누구나 자유롭게 참여할 수 있고 누구나 운영할 수 있기 때문에 공공 블록체인 혹은 개방형 블록체인이라고 한다. 또한, 누구나 데이터를 읽을 수 있기 때문에 투명성 또한 더 크게 보장이 된다. 블록체인 네트워크에 참여하는 각각의 기기를 노드(node)라고 하는데, 모든 노드들이 참여를 하기 때문에 각 거래 내용이 모두에게 공개되며 각 노드들은 블록체인에 저장된 정보들을 복사하여 가지고 있을 수 있고 해시 연산을 수행하여 새로운 블록을 생성할 수 있다. 하지만 검증 또한 모든 노드들이 참여하기 때문에 성능이 저하될 수 있다. 또한, 퍼블릭 블록체인에서는 흔히 코인(Coin)이라

고 불리는 내부 화폐가 필요하다. 분산합의 후 블록을 생성하기 위해 신뢰할 수 있는 노드를 검증하는 단계가 필요한데, 이러한 작업을 위해서는 수고에 대한 댓가가 필요하기 때문이다. 여기 쓰이는 화폐는 마이닝(Mining)이라고 불리는 채굴과정을 통해 획득할 수 있다. 이러한 퍼블릭 블록체인은 네트워크에 참여하는 것이 자유롭듯, 노드의 시스템이 블록체인을 이용하는데 환경적으로 제한이 된다면 네트워크에 연결하지 않고 접속차단으로 이용하지 않는 것도 자유롭다.

2.1.1 프라이빗(private) 블록체인

프라이빗(private) 블록체인은 퍼블릭(public) 블록체인과는 다르게 제한된 인원들만 참여할 수 있는 폐쇄형 블록체인이다. 폐쇄형, 허가형 블록체인이라고 할 수 있다. 프라이빗 블록체인은 관계자의 승인을 얻어야만 참여가 가능하기 때문에 익명성이 보장되지 않는다. 이 특징은 익명성으로 인해 발생하는 문제를 줄일 수 있으며 노드간 권한을 다르게 설정할 수 있고, 허가받은 대상들만 노드로 참여할 수 있기 때문에 퍼블릭 블록체인에 비해 상대적으로 적은 노드수를 운영하게 되어 성능 향상으로 이어진다. 또한, 프라이빗 블록체인의 운영은 암호화폐가 굳이 필요하지 않지만 발행할 수는 있다. 물론 블록체인 네트워크 인원이 제한되어 있으므로 암호화폐의 사용도 내부적으로만 가능하다.

이러한 프라이빗 블록체인의 특징을 기반으로하여 구현한 출입통제 시스템을 이해하기 위해서는 블록체인에 들어간 몇 가지 핵심기술들을 이해해야만 한다.

2.2 P2P(Peer-to-Peer) 네트워크

블록체인 네트워크는 P2P(Peer-to-Peer) 네트워크이기 때문에 Client-Server를 채택하지 않는 탈중앙화 방식이기 때문에 노드들끼리 서로 직접 연결되어있다. 각 노드는 다른 여러 노드들과 상호작용 해야하며, 다른 노드의 상태를 요청하고 자신의 상태와 비교해 오래된 값이면 상태를 업데이트 한다.

2.3 PBFT(Practical Byzantine Fault Tolerance)합의 알고리즘

2.3.1 합의 알고리즘

합의 알고리즘은 블록체인과 같은 P2P 시스템에서 각 노드(Node)간 정보 도달의 시간 차이가 있을 때 참가자들이 하나의 결과에 대한 합의를 얻기 위한 알고리즘이다. 블록체인에서는 분산화된 시스템의 무결성과 보안을 유지시키는 역할을 한다. 또한, 생성된 블록의 정당성을 검토하고 그 블록을 전체 블록체인에 반영하기 위해서 합의 알고리즘이 사용된다. 가장 일반적인 합의 알고리즘은 PoW(Proof of Work: 작업증명)과 PoS(Proof of Stake: 지분증명)이다.

현 프로젝트는 private 블록체인 형태를 가지기 때문에 private 블록체인에서 보편적으로 사용되는 PBFT 합의 알고리즘을 선택하였다. PBFT 합의 알고리즘의 설명은 아래와 같다.

2.3.2 PBFT(Practical Byzantine Fault Tolerance) 합의 알고리즘

PBFT(Practical Byzantine Fault Tolerance) 합의 알고리즘의 이해도를 높이기 위해 먼저 비잔틴 장군 문제에 대해 설명하고자 한다. 비잔틴 장군 문제란 n 개의 비잔틴 부대가 적의 도시를 포위하고 있고, 각 부대는 부대마다 배치된 장군의 명령에 따른다. 한명의 장군은 나머지 $n-1$ 명의 장군과 통신할 때 각각의 장군에게 전령을 보내는 것으로만 통신할 수 있다. 장군들은 지금 총 공격을 할지, 조금 더 기다릴지 합의하여야 한다. 그러나 장군들 중 배신자가 있을 수 있고, 배신자들은 근거 없이 아무 의견이나 제시할 수 있다. 즉, 배신한 장군들의 방해로 공격 여부를 합의할 방법이 필요하게 된다. 이러한 비잔틴 노드가 존재할 수 있는 비동기 시스템일 때 해당 분산시스템에 참여한 모든 노드가 성공적으로 합의를 이룰 수 있도록 개발된 합의 알고리즘이 PBFT 합의 알고리즘이다. PBFT 합의 알고리즘은 네트워크의 모든 참여자를 미리 알고있어야 한다. 참가자 1명이 Primary(또는 Leader)가 되고 자신을 포함한 모든 참가자에게 요청을 보낸다. 그 요청에 대한 결과를 집계한 뒤 다수의 값을 사용해 블록을 확정한다. 부정확한 노드 수를 n 개라고 하면 노드 수는 $3n+1$ 개여야 하며, 확정에는 $n+1$ 개 이상의 노드가 필요하다.

합의는 다음과 같이 수행한다. 1) 우선 클라이언트가 모든 노드에 요청을 브로드캐스트한다. 2) Primary가 순차적으로 명령을 다른 노드에게 전달한다. 각 노드는 브로드캐스트된 명령을 받게 되면 Primary를 포함한 모든 노드에 회신을 한다. 각 노드는 전달된 명령을 일정 수 이상($2n$) 이상 수신하면 Primary를 포함한 모든 노드에 수신한 신호를 재전송한다. 3) 각 노드는 수신된 명령을 일정 수 이상($2n$) 수신하면 명령을 실행하고 블록을 등록해 클라이언트에 Replay된 메시지를 반환한다. 이 과정이 끝나면 모든 노드들은 합의를 이룬 같은 데이터를 가지게 된다.

PoW(Proof of Work: 작업증명)나 PoS(Proof of Stake: 지분증명)와 달리 다수결로 의사결정한 뒤 블록을 만들기 때문에 블록체인의 분기가 발생하지 않는다. 따라서 한 번 확정된 블록은 변경되지 않기 때문에 파이널리티를 확보할 수 있다. 또한 PoW와 같이 조건을 만족시킬 때까지 계산을 반복하지 않아도 되기 때문에 고속으로 동작한다.

부정 사용을 하고자 해도 과반수를 획득해야 하며 만약 Primary가 거짓말을 한다 해도 모든 참가자가 Primary의 움직임을 감시해 거짓말이라고 판단한다면 다수결로 리더 교체를 신청할 수 있기 때문에 장애에 매우 강력한 내성을 지닌 알고리즘이다.

반면 언제나 참가자 전원과 의사소통을 해야 하기 때문에 참가자가 증가하면 통신량이 증가하고 처리량은 저하된다.

2.4 Hash

해시는 '어떤 데이터를 고정된 길이의 데이터로 변환'하는 것을 의미한다. 해시 함수를 거치면 원본 데이터를 알아볼 수 없도록 특수한 문자열로 변환이 된다. 해시 함수는 단방향 변환이기 때문에 해시값을 이용해서 원본 데이터를 복원할 수 없다.

블록체인에서는 이 해시값을 이용해 해당 블록을 서명한다. 또한, 이전 블록의 해시값을 다음 블록에 기록함으로써 체인 형태의 연결 리스트(Linked List)를 형성한다. 특정 블록을 해킹하려면 그 블록에 연결된 다른 블록들도 수정을 해야 하기 때문에 데이터의 위변조가 아주 어렵다.

2.5 인증서

많은 사람들이 네트워크를 통해 패킷을 주고받는다. 이때 여러 공격자로부터 위협이나 민감한 정보를 제 3자가 열람하는 상황을 미연에 방지하기 위한 많은 방법 중 하나이다. 만약 네트워크 데이터가 암호화된다면, 중간에 공격자 패킷을 열람하더라도 데이터가 유출되는 것을 막을 수 있다. 오늘 날 가장 널리 쓰이는 암호화 방식이 SSL/TLS1라는 것인데, 이 방식은 '인증서'라고 하는 일종의 서명을 용한다. 인증서 안에는 다양한 내용이 저장되어 있으며 일반적으로는 인증서의 소유자 이름, 인증서 소유자의 공개 키, 인증서의 유효기간, 인증서이 기타 모든 값들을 해시화한 값 등이 있다. 즉, '당신이 신뢰할 수 있는 사람이나' 라는 것을 확인하기 위한 용도이다.

최상위 인증서를 이용해 하위 인증기관들의 인증서를 발부 할 수 있다. 최상위 인증서는 세상 사람들이 모두 신뢰하기로 약속한 기관에서 발행한 인증서이다. 최상위 인증서가 가지고 있는 비밀키를 이용해 하위 인증기관의 인증서를 복호화 했을 때 성공했다는 의미는 하위 인증서의 내용물을 최상위 인증서의 비밀키로 암호화 해준 것이므로 하위 인증서 또한 신뢰할 수 있다고 간주하는 것이다. 이러한 원리를 Chain of Trust라고 부른다. 상위 인증서가 신뢰할 수 있는 기관이라면, 해당 인증서의 비밀 키로 암호화된 하위 인증서도 신뢰 가능하다고 간주한다. 이러한 인증서의 비밀키와 공개키를 생성하기 위해서는 암호 알고리즘이 필요하다. 현 프로젝트에서는 타원곡선 알고리즘(ECC, Elliptic Curve Cryptosystem)을 이용해 생성했다. ECC는 RSA암호방식에 대한 대안으로 1985년도에 제안된 방식이다. 암호키의 길이가 길면 보안은 강화 되지만, 암호연산 속도가 느려진다. 따라서 보안을 강화하기 위해 RSA방식으로 암호키 길이를 늘리는 방법 대신, ECC 방식을 사용하여야. 즉, ECC를 사용하면, 적은 bit수의 암호키로 동일한 암호성능을 나타내기 때문이다. bit수가 적으면 연산이 보다 빠르게 처리될 수 있기 때문에 암호연산 성능이 좋아지고 암호 연산 장치의 CPU성능이 낮아도, 암호성능을 유지할 수 있다.

2.5.1 인증기관 CA (Certification Authority, root CA)

최상위의 인증기관, 인증 관리 센터. 개인 및 단체 등 모든 개체로부터 신뢰받는 인증기관으로 다른 인증기관에 대한 인증서를 발부한다.

2.5.2 인증기관 ICA (Intermediate Certification Authority)

사용자들에게 인증서를 발급하는 기관. 인증을 통해서 거래의 신뢰를 확보하기 위해 인증서 소지자의 신원을 증명하는 역할을 하며, 이외에도 인증서의 발급 및 인증서의 추출, 폐기, 갱신, 교체 등의 인증업무를 처리하기 위한 전반적인 업무를 한다.

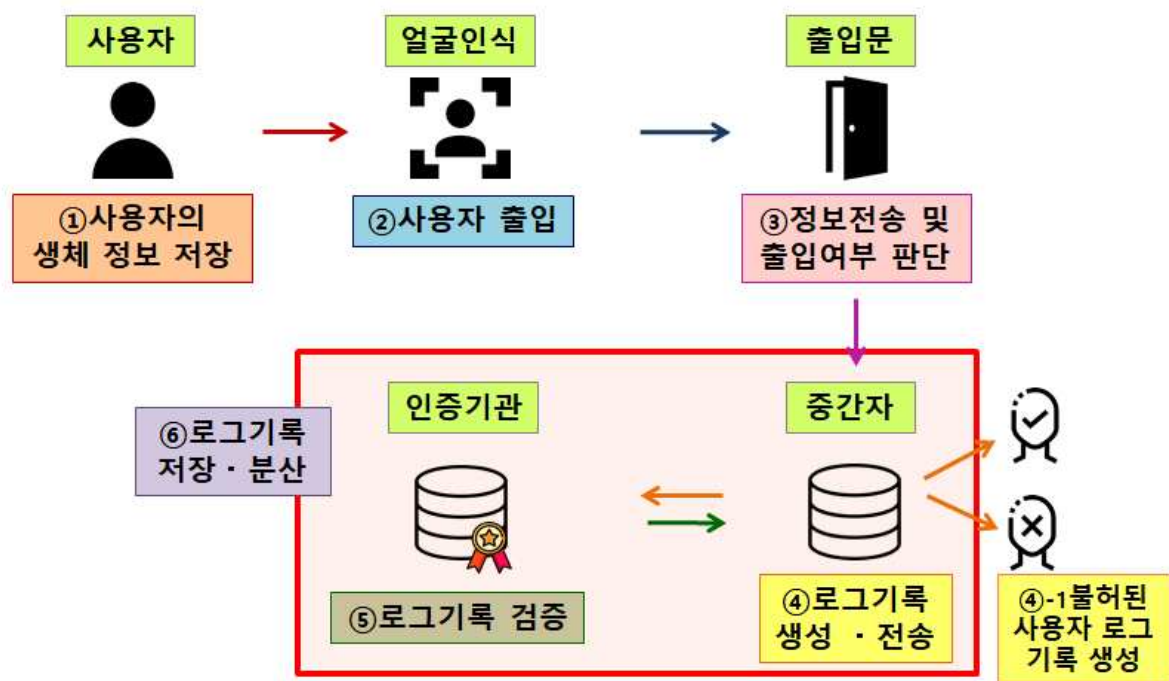
2.6 얼굴인식

얼굴인식 과정에서 첫 번째는 얼굴 검출이다. 이미지에서 얼굴을 찾기 위해서는 이미지를 흑백으로 바꾸는 것부터 시작한다. 얼굴을 찾는데 색상 데이터는 필요없기 때문이다. 이미지에서 얼굴을 분리해낸다음 모든 얼굴에 존재하는 68개의 랜드마크라 부르는 특정

포인트(턱의 상단, 눈 바깥의 가장자리, 눈썹 안쪽의 가장자리 등)를 찾아낸다. 현 프로젝트에서는 이 랜드마크의 벡터 값을 저장하고 실시간으로 카메라로 들어오는 얼굴의 벡터 값을 비교하여 사용자를 판단한다.

3. 본론

3.1 시스템 구성



사용자의 얼굴 정보를 저장한 후 사용자의 출입이 이루어진다. 이때 사용자 얼굴 정보가 알려진 정보이면 출입을 허가하고 출입자의 이름과 출입 시간 정보를 중간자에게 전송한다. 만약, 등록되지 않은 사용자의 정보가 발생할 경우 출입을 허가하지 않으며 불허된 사용자는 "Unknown"으로 처리되며 출입 시간 정보를 중간자에게 전송한다. 이렇게 전달된 정보는 중간자에 의해 로그기록이 생성 및 전송되어진다. 생성된 로그기록은 인증기관에 의해 검증되어지면 저장 및 분산이 진행된다.

3.2 프로그램 구성

3.2.1 인증체계

인증기간 CA를 실행 시 서명을 생성한 후, 각 ICA와 연결되면 생성한 서명을 전송한다.



[그림1]



[그림2]

ICA2
 [두번째 중간인증기관(ICA2)-인증기관(CA) 연결] 연결됨('127.0.0.1', 3000)
 [인증기관(CA) 수신] 연결완료. 데이터 수신 대기중 ...
 [두번째 중간인증기관 서버] 서버 실행 (2002)
 [두번째 중간인증기관 서버] 연결대기 ...
 [인증기관(CA) 수신] 받은정보 : {'type': 'sign_copy', 'cp_name': 'ICA1', 'cp_sign': b"*?Wxd5?;?"
 <eqWxa2Wx07Wx88Wxf4CDWx9c`Wxc9YWx03GWx9aWxc0n3VnWxb9Wxc8Wxd9Wxc8Wxe7'nWxd2Wx00LWxcdi~Wx91Wxf5aqGWx07Wxd1Wxd2copy"}
 [그림3]

ICA3
 [세번째 중간인증기관(ICA3)-인증기관(CA) 연결] 연결됨('127.0.0.1', 3000)
 [인증기관(CA) 수신] 연결완료. 데이터 수신 대기중 ...
 [세번째 중간인증기관 서버] 서버 실행 (2003)
 [세번째 중간인증기관 서버] 연결대기 ...
 [인증기관(CA) 수신] 받은정보 : {'type': 'sign_copy', 'cp_name': 'ICA1', 'cp_sign': b"*?Wxd5?;?"
 <eqWxa2Wx07Wx88Wxf4CDWx9c`Wxc9YWx03GWx9aWxc0n3VnWxb9Wxc8Wxd9Wxc8Wxe7'nWxd2Wx00LWxcdi~Wx91Wxf5aqGWx07Wxd1Wxd2copy"}
 [그림4]

3.2.2 블록체인 생성과정

수신된 사용자 정보가 담긴 트랜잭션을 출입문이 ICA1에 전송하게 되며 ICA1은 ICA2, ICA3과 통신을 통해 합의가 이루어지면 블록이 생성된다.



[그림5]

```
{'number': 2, 'user': 'unknown', 'checkpoint': 'checkpoint1', 'time':
None, 'pass_fail': 'False', 'type': 'tranNsign', 'cp_name': 'ICA1',
'cp_sign': b"*?Wxd5?;?"
<eqWxa2Wx07Wx88Wxf4CDWx9c`Wxc9YWx03GWx9aWxc0n3VnWxb9
Wxc8Wxd9Wxc8Wxe7'nWxd2Wx00LWxcdi~Wx91Wxf5aqGWx07Wxd1W
xd2copy"}
[출입문 수신] 받은정보 : {'type': 'face', 'data': "[('Unknown'",
'passfail': " 'False'", 'time': " '2019-10-1-17:22:23')]"}
```

[그림6]

```
[첫번째 중간인증기관(ICA1) 수신] 받은정보 : {'number': 2, 'user':
'unknown', 'checkpoint': 'checkpoint1', 'time': None, 'pass_fail':
'False', 'type': 'tranNsign', 'cp_name': 'ICA1', 'cp_sign': b"*?Wxd5?;?"
<eqWxa2Wx07Wx88Wxf4CDWx9c`Wxc9YWx03GWx9aWxc0n3VnWxb9
Wxc8Wxd9Wxc8Wxe7'nWxd2Wx00LWxcdi~Wx91Wxf5aqGWx07Wxd1
Wxd2copy"}
```

검증 확인 성공

[그림7]

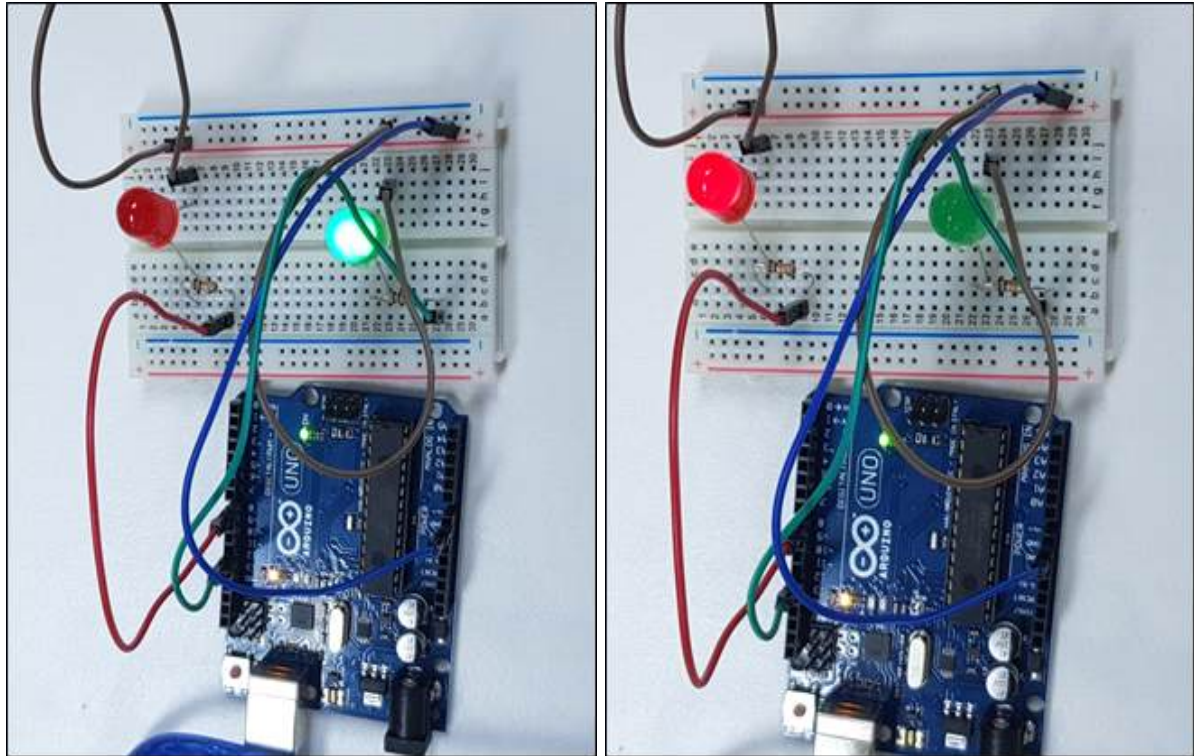
생성된 블록은 인증기관인 CA에서 실시간으로 다음과 같이 확인할 수 있다.



[그림8]

3.2.1 출입 통제

등록된 사용자임이 확인되면 출입을 허용한다. 만약 등록되지 않은 사용자가 출입을 시도할 경우 출입이 허용되지 않는다.



[그림9]

4. 결론

인증된 사용자가 카드키 출입이 아닌 얼굴인식을 사용하여 편리성과 등록되지 않은 사용자의 출입을 방지함으로 위험요소를 1차적으로 방지하며 블록체인을 기반으로 출입 통제 시스템을 만듦으로써 출입 시 발생한 로그기록이 위변조 되는 것을 방지할 수 있다.

5. 별첨

5-1. 참고자료

5.1.1 소스코드

5.1.2 발표자료

5.1.1 소스코드

bc.py

```
import hashlib
from datetime import datetime
import random
import time
class Blockchain(object):
    def __init__(self):
        self.chain = []
        self.current_transactions = []
        self.vote = []
        self.last_block={}
        self.new_block(previous_hash = 1)

    def new_block(self, previous_hash, nonce=None,
                  difficulty=None, merkleroot=None):

        now = datetime.now()
        t = now.strftime('%Y.%m.%d.%H:%M')
        block = {
            'type' : 'block',
            'index' : len(self.chain) + 1,
            'timestamp' : t,
            'transactions' : self.current_transactions,
            'previous_hash' :previous_hash ,
            'merkleroot' : merkleroot,
        }

        self.current_transactions = []
        self.last_block=block
        return block

    def new_transaction(self, number, user, checkpoint, time, pass_fail):
        self.current_transactions.append({
            'type' : 'transaction',
            'number' : len(self.current_transactions)+1,
            'user': user,
            'checkpoint' : checkpoint,
            'time' : time,
            'pass_fail' : pass_fail,
        })

        return self.last_block['index'] + 1

    def hash(block):
        block_string = json.dumps(block, sort_keys = True).encode()
        return hashlib.sha256(block_string).hexdigest()

    def hash_block(self):
        sha = hashlib.sha256()
        hash_string = (str(self.chain[-1]['index'])+str(self.chain[-1]['timestamp'])
+str(self.chain[-1]['transactions'])+str(self.chain[-1]['previous_hash'])
+str(self.chain[-1]['merkleroot']))
        sha.update(hash_string.encode())
        return sha.hexdigest()
```

```

def input_transaction(self,dic):
    del dic['type']
    self.new_transaction(dic['number'],
                        dic['user'], dic['checkpoint'],dic['time'], dic['pass_fail'])

def last_block(self):
    return self.chain[-1]

def last_tran(self):
    return self.current_transactions[-1]

def print_transaction(self):
    print(self.chain[1][1]['transactions'][0][1]['user'])

def print_block(self,start_point = -1, finish_point = 0):
    if start_point == -1:
        print("블록번호 : "+str(self.last_block[1]['index'])+
              "\n발생시간 : "+str(self.last_block[1]['timestamp'])+
              "\n이전해쉬 : "+str(self.last_block[1]['previous_hash'])+
              "\n현재해쉬 : "+str(self.hash_block())+
              "\n머클루트 : "+str(self.last_block[1]['merkleroot']))
        self.print_transaction(self.last_block[1]['transactions'],self.last_block[1]['index'])
    elif start_point > -1 and finish_point == 0:
        print("블록번호 : "+str(self.chain[start_point][1]['index'])+
              "\n발생시간 : "+str(self.chain[start_point][1]['timestamp'])+
              "\n이전해쉬 : "+str(self.chain[start_point][1]['previous_hash'])+
              "\n현재해쉬 : "+str(self.hash_block())+
              "\n머클루트 : "+str(self.chain[start_point][1]['merkleroot']))
        self.print_transaction(self.chain[start_point][1]['transactions'],start_point)
    elif start_point > -1 and finish_point > -1:
        for i in range(start_point, finish_point):
            print("블록번호 : "+str(self.chain[i][1]['index'])+
                  "\n발생시간 : "+str(self.chain[i][1]['timestamp'])+
                  "\n이전해쉬 : "+str(self.chain[i][1]['previous_hash'])+
                  "\n현재해쉬 : "+str(self.hash_block())+
                  "\n머클루트 : "+str(self.chain[i][1]['merkleroot']))
            self.print_transaction(self.chain[i][1]['transactions'],i)
        print("\n\n\n")

def print_transaction(self,transaction,index):
    if len(transaction)!=0 and index!=0:
        print("====="+str(index+1)+"번 블록의 트랜잭션 시작=====\n")
        for i in range(len(transaction)):
            print("트랜잭션번호 : "+str(transaction[i][1]['number'])+
                  "\n발생시간 : "+str(transaction[i][1]['time'])+
                  "\n출입자 : "+str(transaction[i][1]['user'])+
                  "\n체크포인트 : "+str(transaction[i][1]['checkpoint'])+
                  "\n개폐여부 : "+str(transaction[i][1]['pass_fail'])+"\n")
        print("====="+str(index+1)+"번 블록의 트랜잭션 끝=====\n")
    else:
        print("트랜잭션 : None")

def new_user(self):
    pass

def vote_block(self, vote,userinfo):

```



```

agree = 0
for k in range(10):
    for i in range(5):
        i = random.randrange(0,2)
        vote.append(i)
    print(vote)
    for j in range(5):
        if vote[j] == 1:
            agree += 1
    if agree >= (len(vote)*0.51):
        userinfo['pass_fail'] = "PASS"
        self.input_transaction(userinfo)
        self.new_block(self.hash_block(),2,3,4)
        self.print_block()
        print("블록생성 성공")
    else:
        userinfo['pass_fail'] = "FAIL"
        self.input_transaction(userinfo)
        print("블록생성 실패")
    vote = [0]*5
    agree = 0
    time.sleep(1)

```

node.py

```

import zmq
import json

class Node():
    def __init__(self):
        self.socks = []
        self.ctx = zmq.Context()

    def init_server(self, port):
        self.server_sock = self.ctx.socket(zmq.REQ)
        self.server_sock.bind('tcp://*:%s' % port)

    def connect_node(self, other_port):
        self.client_sock = self.ctx.socket(zmq.REP)
        self.client_sock.connect('tcp://localhost:%s' % other_port)

    def send_server(self, data):
        msg = json.dumps(data, separators=(',', ':'))
        self.server_sock.send_string(msg)

    def send_client(self, data):
        msg = json.dumps(data, separators=(',', ':'))
        self.client_sock.send_string(msg)

    def recv_server(self):
        return self.server_sock.recv()

    def recv_client(self):
        return self.client_sock.recv()

    def recv_client(self):
        return self.client_sock.recv()

```

```

import time
from datetime import datetime
import json
import random
import hashlib
import threading
from socket import *
from ecdsa import SigningKey
import pickle
import datetime as d
from bc import Blockchain

from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QMessageBox
import sys
import sec

class Ui_Main_F(QtWidgets.QWidget):
    def __init__(self):
        super().__init__()
        self.Popups = []
        self.count=0
        self.checkBox__ = []
        self.label__ = []
    def setupUi(self, Main_F):
        Main_F.setObjectName("Main_F")
        Main_F.resize(969, 678)
        Main_F.setStyleSheet("background-color: rgb(255,254,239);")
        self.label = QtWidgets.QLabel(Main_F)
        self.label.setGeometry(QtCore.QRect(0, 0, 969, 51))
        font = QtGui.QFont()
        font.setFamily("맑은 고딕")
        font.setPointSize(16)
        font.setBold(True)
        font.setWeight(75)
        self.label.setFont(font)
        self.label.setStyleSheet("background-color: rgb(170, 0, 0);")
        self.label.setFrameShape(QtWidgets.QFrame.NoFrame)
        self.label.setObjectName("label")
        self.tabWidget = QtWidgets.QTabWidget(Main_F)
        self.tabWidget.setGeometry(QtCore.QRect(40, 100, 900, 530))
        font = QtGui.QFont()
        font.setFamily("맑은 고딕")
        font.setPointSize(14)
        font.setBold(True)
        font.setWeight(75)
        self.tabWidget.setFont(font)
        self.tabWidget.setStyleSheet("")
        self.tabWidget.setObjectName("tabWidget")
        self.tab = QtWidgets.QWidget()
        self.tab.setObjectName("tab")

```

```

self.listView_3 = QtWidgets.QListView(self.tab)
self.listView_3.setGeometry(QtCore.QRect(0, 0, 901, 501))
self.listView_3.setStyleSheet("background-color: rgb(255, 255, 255);\n"
"border-top: 5px solid #002060;")
self.listView_3.setLineWidth(1)
self.listView_3.setMidLineWidth(0)
self.listView_3.setObjectName("listView_3")
self.label_2 = QtWidgets.QLabel(self.tab)
self.label_2.setGeometry(QtCore.QRect(470, 17, 141, 31))
font = QtGui.QFont()
font.setFamily("맑은 고딕")
font.setBold(True)
font.setWeight(75)
self.label_2.setFont(font)
self.label_2.setStyleSheet("color: rgb(111, 111, 111);")
self.label_2.setObjectName("label_2")
self.label_3 = QtWidgets.QLabel(self.tab)
self.label_3.setGeometry(QtCore.QRect(590, 17, 51, 31))
font = QtGui.QFont()
font.setFamily("맑은 고딕")
font.setBold(True)
font.setWeight(75)
self.label_3.setFont(font)
self.label_3.setStyleSheet("color: rgb(111, 111, 111);")
self.label_3.setObjectName("label_3")
self.pushButton = QtWidgets.QPushButton(self.tab)
self.pushButton.setGeometry(QtCore.QRect(680, 21, 75, 23))
font = QtGui.QFont()
font.setFamily("맑은 고딕")
font.setBold(True)
font.setWeight(75)
self.pushButton.setFont(font)
self.pushButton.setStyleSheet("background-color: rgb(52, 99, 255);\n"
"color: rgb(255, 255, 255);\n"
"border-top: 0px solid #002060;")
self.pushButton.setObjectName("pushButton")
self.pushButton_2 = QtWidgets.QPushButton(self.tab)
self.pushButton_2.setGeometry(QtCore.QRect(780, 21, 75, 23))
font = QtGui.QFont()
font.setFamily("맑은 고딕")
font.setBold(True)
font.setWeight(75)
self.pushButton_2.setFont(font)
self.pushButton_2.setStyleSheet("background-color: rgb(61, 61, 61);\n"
"background-color: rgb(84, 84, 84);\n"
"color: rgb(255, 255, 255);\n"
"border-top: 0px solid #002060;")
self.pushButton_2.setObjectName("pushButton_2")
self.checkBox = QtWidgets.QCheckBox(self.tab)
self.checkBox.setGeometry(QtCore.QRect(27, 73, 16, 16))
self.checkBox.setStyleSheet("background-color: rgb(250,254,245);")
self.checkBox.setText("")
self.checkBox.setObjectName("checkBox")
self.listView = QtWidgets.QListView(self.tab)
self.listView.setGeometry(QtCore.QRect(-10, 60, 911, 41))
self.listView.setStyleSheet("border-top: 2px solid #9e9e9e;\n"
"border-bottom: 2px solid #9e9e9e;\n"

```



```

"background-color: rgb(250,254,245);")
    self.listView.setObjectName("listView")
    self.label_4 = QtWidgets.QLabel(self.tab)
    self.label_4.setGeometry(QtCore.QRect(100, 70, 71, 21))
    font = QtGui.QFont()
    font.setFamily("맑은 고딕")
    font.setPointSize(10)
    font.setBold(True)
    font.setWeight(75)
    self.label_4.setFont(font)
    self.label_4.setStyleSheet("color: rgb(111, 111, 111);\n"
"background-color: rgb(250,254,245);")
    self.label_4.setObjectName("label_4")
    self.label_5 = QtWidgets.QLabel(self.tab)
    self.label_5.setGeometry(QtCore.QRect(260, 70, 71, 21))
    font = QtGui.QFont()
    font.setFamily("맑은 고딕")
    font.setPointSize(10)
    font.setBold(True)
    font.setWeight(75)
    self.label_5.setFont(font)
    self.label_5.setStyleSheet("color: rgb(111, 111, 111);\n"
"background-color: rgb(250,254,245);")
    self.label_5.setObjectName("label_5")
    self.label_6 = QtWidgets.QLabel(self.tab)
    self.label_6.setGeometry(QtCore.QRect(580, 70, 71, 21))
    font = QtGui.QFont()
    font.setFamily("맑은 고딕")
    font.setPointSize(10)
    font.setBold(True)
    font.setWeight(75)
    self.label_6.setFont(font)
    self.label_6.setStyleSheet("color: rgb(111, 111, 111);\n"
"background-color: rgb(250,254,245);")
    self.label_6.setObjectName("label_6")
    self.label_7 = QtWidgets.QLabel(self.tab)
    self.label_7.setGeometry(QtCore.QRect(420, 70, 71, 21))
    font = QtGui.QFont()
    font.setFamily("맑은 고딕")
    font.setPointSize(10)
    font.setBold(True)
    font.setWeight(75)
    self.label_7.setFont(font)
    self.label_7.setStyleSheet("color: rgb(111, 111, 111);\n"
"background-color: rgb(250,254,245);")
    self.label_7.setObjectName("label_7")
    self.label_8 = QtWidgets.QLabel(self.tab)
    self.label_8.setGeometry(QtCore.QRect(740, 70, 101, 21))
    font = QtGui.QFont()
    font.setFamily("맑은 고딕")
    font.setPointSize(10)
    font.setBold(True)
    font.setWeight(75)
    self.label_8.setFont(font)
    self.label_8.setStyleSheet("\n"
"color: rgb(111, 111, 111);\n"
"background-color: rgb(250,254,245);")

```

```

        self.label_8.setObjectName("label_8")
        self.listView_4 = QtWidgets.QListView(self.tab)
        self.listView_4.setGeometry(QtCore.QRect(-20, 100, 921, 41))
        self.listView_4.setStyleSheet("border: 0px solid #9e9e9e;\n"
"border-bottom: 1px solid #9e9e9e;")
        self.listView_4.setObjectName("listView_4")
        self.listView_5 = QtWidgets.QListView(self.tab)
        self.listView_5.setGeometry(QtCore.QRect(-20, 145, 921, 41))
        self.listView_5.setStyleSheet("border: 0px solid #9e9e9e;\n"
"border-bottom: 1px solid #9e9e9e;")
        self.listView_5.setObjectName("listView_5")
        self.listView_6 = QtWidgets.QListView(self.tab)
        self.listView_6.setGeometry(QtCore.QRect(-20, 190, 921, 41))
        self.listView_6.setStyleSheet("border: 0px solid #9e9e9e;\n"
"border-bottom: 1px solid #9e9e9e;")
        self.listView_6.setObjectName("listView_6")
        self.listView_7 = QtWidgets.QListView(self.tab)
        self.listView_7.setGeometry(QtCore.QRect(-20, 235, 921, 41))
        self.listView_7.setStyleSheet("border: 0px solid #9e9e9e;\n"
"border-bottom: 1px solid #9e9e9e;")
        self.listView_7.setObjectName("listView_7")
        self.listView_8 = QtWidgets.QListView(self.tab)
        self.listView_8.setGeometry(QtCore.QRect(-20, 280, 921, 41))
        self.listView_8.setStyleSheet("border: 0px solid #9e9e9e;\n"
"border-bottom: 1px solid #9e9e9e;")
        self.listView_8.setObjectName("listView_8")
        self.listView_9 = QtWidgets.QListView(self.tab)
        self.listView_9.setGeometry(QtCore.QRect(-20, 325, 921, 41))
        self.listView_9.setStyleSheet("border: 0px solid #9e9e9e;\n"
"border-bottom: 1px solid #9e9e9e;")
        self.listView_9.setObjectName("listView_9")
        self.listView_10 = QtWidgets.QListView(self.tab)
        self.listView_10.setGeometry(QtCore.QRect(-20, 370, 921, 41))
        self.listView_10.setStyleSheet("border: 0px solid #9e9e9e;\n"
"border-bottom: 1px solid #9e9e9e;")

        self.listView_10.setObjectName("listView_10")
        self.pushButton_3 = QtWidgets.QPushButton(self.tab)
        self.pushButton_3.setGeometry(QtCore.QRect(440, 440, 21, 23))
        font = QtGui.QFont()
        font.setFamily("맑은 고딕")
        font.setBold(False)
        font.setWeight(50)
        self.pushButton_3.setFont(font)
        self.pushButton_3.setStyleSheet("background-color: rgb(255, 255, 255);\n"
"border: 0px solid #9e9e9e;")
        self.pushButton_3.setObjectName("pushButton_3")
        self.pushButton_4 = QtWidgets.QPushButton(self.tab)
        self.pushButton_4.setGeometry(QtCore.QRect(460, 440, 21, 23))
        font = QtGui.QFont()
        font.setFamily("맑은 고딕")
        font.setBold(True)
        font.setWeight(75)
        self.pushButton_4.setFont(font)
        self.pushButton_4.setStyleSheet("background-color: rgb(255, 255, 255);\n"
"border: 0px solid #9e9e9e;")
        self.pushButton_4.setObjectName("pushButton_4")

```

```

self.pushButton_5 = QtWidgets.QPushButton(self.tab)
self.pushButton_5.setGeometry(QtCore.QRect(420, 440, 21, 23))
font = QtGui.QFont()
font.setFamily("맑은 고딕")
font.setBold(True)
font.setWeight(75)
self.pushButton_5.setFont(font)
self.pushButton_5.setStyleSheet("background-color: rgb(255, 255, 255);\n"
"border: 0px solid #9e9e9e;")
self.pushButton_5.setObjectName("pushButton_5")
self.listView_3.raise_()
self.label_3.raise_()
self.label_2.raise_()
self.pushButton.raise_()
self.pushButton_2.raise_()
self.listView.raise_()
self.label_4.raise_()
self.label_5.raise_()
self.label_6.raise_()
self.label_7.raise_()
self.label_8.raise_()
self.checkBox.raise_()
self.listView_4.raise_()
self.listView_5.raise_()
self.listView_6.raise_()
self.listView_7.raise_()
self.listView_8.raise_()
self.listView_9.raise_()
self.listView_10.raise_()
self.pushButton_3.raise_()
self.pushButton_4.raise_()
self.pushButton_5.raise_()
self.tabWidget.addTab(self.tab, "")
self.tab_3 = QtWidgets.QWidget()
self.tab_3.setObjectName("tab_3")
self.plainTextEdit = QtWidgets.QPlainTextEdit(self.tab_3)
self.plainTextEdit.setGeometry(QtCore.QRect(0, 0, 891, 491))
font = QtGui.QFont()
font.setFamily("맑은 고딕")
font.setPointSize(14)
self.plainTextEdit.setFont(font)
self.plainTextEdit.setStyleSheet("background-color: rgb(255, 255, 255);\n"
"border: 3px solid #002060;")
self.plainTextEdit.setReadOnly(True)
self.plainTextEdit.setObjectName("plainTextEdit")
self.tabWidget.addTab(self.tab_3, "")
self.tab_2 = QtWidgets.QWidget()
self.tab_2.setObjectName("tab_2")
self.plainTextEdit_2 = QtWidgets.QPlainTextEdit(self.tab_2)
self.plainTextEdit_2.setGeometry(QtCore.QRect(0, 0, 891, 491))
font = QtGui.QFont()
font.setFamily("맑은 고딕")
font.setPointSize(14)
self.plainTextEdit_2.setFont(font)
self.plainTextEdit_2.setStyleSheet("background-color: rgb(255, 255, 255);\n"
"border: 3px solid #002060;")
self.plainTextEdit_2.setReadOnly(True)

```

```

self.plainTextEdit_2.setPlainText("")
self.plainTextEdit_2.setObjectName("plainTextEdit_2")
self.tabWidget.addTab(self.tab_2, "")

for i in range(7):
    self.checkBox_2.append(QtWidgets.QCheckBox(self.tab))
    self.checkBox_2[self.count].setGeometry(QtCore.QRect(27,
112+(self.count*45), 16, 16))
    self.checkBox_2[self.count].setStyleSheet("background-color:
rgb(250,254,245);")
    self.checkBox_2[self.count].setText("")
    self.checkBox_2[self.count].setObjectName("checkBox_2")
    self.checkBox_2[self.count].raise_()
    self.checkBox_2[self.count].hide()

    font = QtGui.QFont()
    font.setFamily("맑은 고딕")

    self.label_2.append(QtWidgets.QLabel(self.tab))
    self.label_2[-1].setGeometry(QtCore.QRect(115, 112+(self.count*45), 41, 16))
    self.label_2[-1].setFont(font)
    self.label_2[-1].setObjectName("label"+str(self.count))
    self.label_2[-1].raise_()

    self.label_2.append(QtWidgets.QLabel(self.tab))
    self.label_2[-1].setGeometry(QtCore.QRect(245, 105+(self.count*45), 91, 31))
    self.label_2[-1].setFont(font)
    self.label_2[-1].setObjectName("label1_"+str(self.count))
    self.label_2[-1].raise_()

    self.label_2.append(QtWidgets.QLabel(self.tab))
    self.label_2[-1].setGeometry(QtCore.QRect(435, 105+(self.count*45), 31, 31))
    self.label_2[-1].setFont(font)
    self.label_2[-1].setObjectName("label2_"+str(self.count))
    self.label_2[-1].raise_()

    self.label_2.append(QtWidgets.QLabel(self.tab))
    self.label_2[-1].setGeometry(QtCore.QRect(560, 105+(self.count*45), 131,
31))
    self.label_2[-1].setFont(font)
    self.label_2[-1].setObjectName("label3_"+str(self.count))
    self.label_2[-1].raise_()

    self.label_2.append(QtWidgets.QLabel(self.tab))
    self.label_2[-1].setGeometry(QtCore.QRect(740, 105+(self.count*45), 101,
31))
    self.label_2[-1].setFont(font)
    self.label_2[-1].setObjectName("label4_"+str(self.count))
    self.label_2[-1].raise_()

    self.count = self.count+1
self.count=0

```

```

self.pushButton.clicked.connect(self.B1_C)
self.pushButton_2.clicked.connect(self.B2_C)

self.retranslateUi(Main_F)
self.tabWidget.setCurrentIndex(0)
QtCore.QMetaObject.connectSlotsByName(Main_F)

def B1_C(self):
    self.Popups.append(sec.Ui_Form(self))
    self.Popups[-1].show()

def B2_C(self):
    reply = QMessageBox.question(self, '사용자 삭제',
                                "선택한 사용자를 삭제하시겠습니까?",
                                QMessageBox.Yes |
                                QMessageBox.No, QMessageBox.No)

    if reply == QMessageBox.Yes:
        for i in range(self.count+1):
            if self.checkBox__[i].isChecked():
                self.checkBox__[i].hide()
                self.checkBox__[i].setChecked(False)
                self.label__[(5*i)].setText("")
                self.label__[(5*i)+1].setText("")
                self.label__[(5*i)+2].setText("")
                self.label__[(5*i)+3].setText("")
                self.label__[(5*i)+4].setText("")
                self.label_3.setText("<html><head /><body><p
align=\"center\"><span
font-size:11pt;\">"+str(self.count-1)+"</span></p></body></html>")
                self.count = self.count-1
                for j in range(i-1,self.count):
                    if self.label__[(5*j)+1].text() == "":
                        if j != -1:
                            self.checkBox__[j].show()
                            self.label__[(5*j)].setText(self.label__[(5*(j+1))].text())
                            self.label__[(5*j)+1].setText(self.label__[(5*(j+1))+1].text())
                            self.label__[(5*j)+2].setText(self.label__[(5*(j+1))+2].text())
                            self.label__[(5*j)+3].setText(self.label__[(5*(j+1))+3].text())
                            self.label__[(5*j)+4].setText(self.label__[(5*(j+1))+4].text())

                            self.checkBox__[j+1].hide()
                            self.label__[(5*(j+1))].setText("")
                            self.label__[(5*(j+1))+1].setText("")
                            self.label__[(5*(j+1))+2].setText("")
                            self.label__[(5*(j+1))+3].setText("")
                            self.label__[(5*(j+1))+4].setText("")

            else:

                print()

def settext(self, str_):
    self.plainTextEdit.appendPlainText(str_)

```

```

def settext2(self, str_):
    self.plainTextEdit_2.appendPlainText(str_)

def retranslateUi(self, Main_F):
    _translate = QtCore.QCoreApplication.translate
    Main_F.setWindowTitle(_translate("Main_F", "GUI CA"))
    Main_F.setToolTip(_translate("Main_F",
"<html><head/><body><p>hi</p><p><br/></p></body></html>"))
    Main_F.setWhatsThis(_translate("Main_F",
"<html><head/><body><p>h</p></body></html>"))
    self.label.setText(_translate("Main_F",
align="center"><span style="
color:#ffffff;">Well-made</span></p></body></html>"))
    self.label_2.setText(_translate("Main_F",
align="center"><span style=" font-size:11pt;">등록된 사용자 수
:</span></p></body></html>"))
    self.label_3.setText(_translate("Main_F",
align="center"><span style=" font-size:11pt;">0</span></p></body></html>"))
    self.pushButton.setText(_translate("Main_F", "추가"))
    self.pushButton_2.setText(_translate("Main_F", "삭제"))
    self.label_4.setText(_translate("Main_F", "사용자 이름"))
    self.label_5.setText(_translate("Main_F", "전화번호"))
    self.label_6.setText(_translate("Main_F", "이메일주소"))
    self.label_7.setText(_translate("Main_F", "해당부서"))
    self.label_8.setText(_translate("Main_F", "마지막 출입시간"))
    self.pushButton_3.setText(_translate("Main_F", "1"))
    self.pushButton_4.setText(_translate("Main_F", ">"))
    self.pushButton_5.setText(_translate("Main_F", "<"))
    self.tabWidget.setTabText(self.tabWidget.indexOf(self.tab), _translate("Main_F", "
사용자 관리"))
    self.plainTextEdit.setPlainText(_translate("Main_F", "CA"))
    self.tabWidget.setTabText(self.tabWidget.indexOf(self.tab_3), _translate("Main_F",
"로그 기록"))
    self.tabWidget.setTabText(self.tabWidget.indexOf(self.tab_2), _translate("Main_F",
"실시간 블록"))

app = QtWidgets.QApplication(sys.argv)
Main_F = QtWidgets.QDialog()
ui = Ui_Main_F()
ui.setupUi(Main_F)
Main_F.show()

class Signature():
    def __init__(self):
        self.back_signfile = []
        self.back2_signfile = []
        self.back3_signfile = []
        with open('create_signfile.txt', mode='w', encoding='utf-8') as file:
            file.close()
        with open('verify_signfile.pickle', mode='wb') as file:
            file.close()
        with open('delete_signfile.txt', mode='a', encoding='utf-8') as file:
            file.close()

```

```

def create_sign(self, i):
    global ui
    self.sk = SigningKey.generate()
    self.vk = self.sk.get_verifying_key()
    self.name = "ICA" + str(i)
    self.sign = self.sk.sign(self.name.encode())
    year = d.datetime.today().year
    month = d.datetime.today().month
    day = d.datetime.today().day
    self.time = d.date(year, month, day)
    ui.settext("=====서명 생성 완료=====\\n"+str(self.sk.to_string()))
    ui.settext("=====\\n")

    with open('create_signfile.txt', mode='a', encoding='utf-8') as file:
        file.write("서명 : " + self.name +
                  "\\n암호키 : " + str(self.sk.to_string()) +
                  "\\n확인키 : " + str(self.vk.to_string()) +
                  "\\n전자서명 : " + str(self.sign) +
                  "\\n생성시간 : " + str(self.time) +

                  "\\n=====\\n"
                  )
        file.close()

    with open('verify_signfile.pickle', mode='ab') as file:
        pickle.dump(self.name, file, pickle.HIGHEST_PROTOCOL)
        pickle.dump(self.vk, file, pickle.HIGHEST_PROTOCOL)
        pickle.dump(self.sign, file, pickle.HIGHEST_PROTOCOL)
        file.close()

    return self.name

def append_sign(self, i):
    global ui
    self.app_name = "ICA" + str(i)

    with open('verify_signfile.pickle', mode='rb') as file:
        try:
            while(True):
                line = pickle.load(file)
                if self.app_name == line:
                    ui.settext("이미 있는 서명입니다.\\n")
                    file.close()
                    return
        except EOFError as e:
            ui.settext("새로운 서명을 생성합니다.\\n")
            file.close()
            Sign.create_sign(i)

```

```

def copy_sign(self, cp_name):
    self.cp_name = cp_name
    global ui

    with open('verify_signfile.pickle', mode='rb') as file:
        try :
            while(True):
                line = pickle.load(file)
                if self.cp_name == line:
                    break
            except EOFError as e:

                ui.setText("잘못된 입력입니다.\n")
                file.close()
                return False, False
        file.close()

    with open('verify_signfile.pickle', mode='rb') as file:
        while(True):
            if self.cp_name == pickle.load(file):
                pickle.load(file)
                self.file_sign = pickle.load(file)
                break
        file.close()

    self.cp_sign = bytes(self.file_sign + "copy".encode())

    return self.cp_sign, self.cp_name

def verify_sign(self, vk_sign, vk_name):
    self.vk_sign = vk_sign[:48]
    global ui

    with open('verify_signfile.pickle', mode='rb') as file:
        while(True):
            if vk_name == pickle.load(file):
                self.filevk_sign = pickle.load(file)
                break
        file.close()

    assert self.filevk_sign.verify(self.vk_sign, vk_name.encode())
    try:
        ui.setText("검증 확인 성공\n")
        verify= True
    except Exception as e:
        ui.setText("검증 확인 실패\n")
        verify= False
    return verify

def delete_sign(self, del_name):
    self.del_name = del_name
    global ui

```



```

if del_name == "":

    with open('create_signfile.txt', mode='r',encoding='utf-8') as file:
        i = 0
        while(True):
            line = file.readline()
            i = i+1
            if i%6 == 5:
                a = line.split("-")
                expiry = int(d.datetime.today().day)-int(a[2])
                if expiry == 0:
                    ui.settext(a[2])
                    break
            file.close()

    with open('verify_signfile.pickle', mode='rb') as file:
        try:
            while(True):
                line = pickle.load(file)
                if self.del_name == line:
                    break
        except EOFError as e:
            ui.settext("잘못된 입력입니다.\n")
            file.close()
        return
    file.close()

    with open('create_signfile.txt', mode='r',encoding='utf-8') as file:
        while(True):
            line = file.readline()
            if self.del_name in line:
                for i in range(4):
                    line = file.readline()
                    self.back_signfile.append(line)
                break
            file.close()

    with open('delete_signfile.txt', mode='a', encoding='utf-8') as file:
        file.write("서명 : " + self.del_name + "\n" +
                self.back_signfile[0] +
                self.back_signfile[1] +
                self.back_signfile[2] +
                self.back_signfile[3] +
                "=====\\n"
                )
        file.close()
        ui.settext("delete 생성 성공\\n")

    with open('create_signfile.txt', mode='r',encoding='utf-8') as file:
        while(True):

            line = file.readline()
            if self.del_name in line:

```

```

        for i in range(5):
            line = file.readline()
        else:
            self.back2_signfile.append(line)

        line = file.readline()
        self.back2_signfile.append(line)
        if not line:
            break
    file.close()

    with open('create_signfile.txt', mode='w+', encoding='utf-8') as file:
        for i in range(2):
            file.write(self.back2_signfile[(6*i)] +
                      self.back2_signfile[(6*i)+1] +
                      self.back2_signfile[(6*i)+2] +
                      self.back2_signfile[(6*i)+3] +
                      self.back2_signfile[(6*i)+4] +
                      self.back2_signfile[(6*i)+5]
                      )
        file.close()
        ui.setText("create 삭제 성공\n")

    with open('verify_signfile.pickle', mode='rb') as file:
        try:
            while(True):
                line = pickle.load(file)
                if self.del_name == line:
                    for i in range(2):
                        line = pickle.load(file)
                    else:
                        self.back3_signfile.append(line)
        except EOFError as e:
            file.close()
    with open('verify_signfile.pickle', mode='wb+') as file:
        for i in range(2):
            pickle.dump(self.back3_signfile[(3*i)], file, pickle.HIGHEST_PROTOCOL)
            pickle.dump(self.back3_signfile[(3*i)+1], file,
pickle.HIGHEST_PROTOCOL)
            pickle.dump(self.back3_signfile[(3*i)+2], file,
pickle.HIGHEST_PROTOCOL)
        file.close()
        ui.setText("verify 삭제 성공\n")

class Node(Signature):
    def __init__(self, host, port):
        self.host = host
        self.port = int(port)
        self.nodes = []

    def __debug(self, msg):
        global ui

```

```

ui.setText('[%s] %s' % (str(threading.currentThread().getName()), msg))

def __debug_server(self, msg):
    ui.setText('[인증기관 서버(CA)] %s\n' % (msg))

def __debug_client(self, msg):
    th_name = str(threading.currentThread().getName())
    if th_name == 'Thread-2':
        ui.setText('[첫번째 중간인증기관(ICA1) 수신] %s\n' % (msg))
    elif th_name == 'Thread-3':
        ui.setText('[두번째 중간인증기관(ICA2) 수신] %s\n' % (msg))
    elif th_name == 'Thread-4':
        ui.setText('[세번째 중간인증기관(ICA3) 수신] %s\n' % (msg))

def run_server(self):
    self.s = socket(AF_INET, SOCK_STREAM)
    self.s.bind((self.host, self.port))
    self.s.listen(5)

    self.__debug_server('서버 실행 (%d)' % self.port)
    count_node = 0
    while True:
        self.__debug_server('연결대기 ... ')
        client_sock, client_addr = self.s.accept()
        self.nodes.append(client_sock)
        self.__debug_server('연결됨' + str(self.nodes[-1].getpeername()))
        t_recv = threading.Thread(target = self.recv, args={self.nodes[-1]})
        t_recv.start()

def run_client(self, host, port):
    s = socket(AF_INET, SOCK_STREAM)
    s.connect((host, int(port)))
    self.__debug_client('연결됨' + str(s.getpeername()))
    return s

def __send(self, s, data):
    msg = pickle.dumps(data)
    s.sendall(msg)

def __send_all(self, s, data, flag = 0):
    for node in self.nodes:
        time.sleep(1)
        if(flag == 0):
            if(node != s):
                self.__send(node, data)
        else:
            self.__send(node, data)

def recv(self, s):
    global ui

    ica_num = self.create_sign(1)

    cp_sign, cp_name = self.copy_sign(ica_num)

```

```

self.__send(s, {'type':'sign_copy', 'cp_name':cp_name, 'cp_sign':cp_sign})

self.__debug_client('데이터 수신 대기중 ... ')
count_vote_P = 0
count_vote_F = 0

while True:
    msg = s.recv(1024)
    data = pickle.loads(msg)

    if data:
        if(data.get('type') == 'tranNsign'):

            sign_PF = self.verify_sign(data.get('cp_sign'), data.get('cp_name'))

            if sign_PF:
                del data['cp_name']
                del data['cp_sign']
                ui.setText(str(data))
                self.__send_all(s, data)
            elif(data.get('type') == 'block'):

                ui.setText2("블록번호 : "+str(data['index'])+
                    "\n발생시간 : "+str(data['timestamp'])+
                    "\n이전해쉬 : "+str(data['previous_hash']))
                self.print_transaction(data['transactions'],data['index'])

                self.__send_all(s, {'type':'vote', 'data':sign_PF}, 1)
            elif(data.get('type') == 'r_vote'):
                if (data.get('data')==0):
                    count_vote_F +=1

                else:
                    count_vote_P +=1
                count = count_vote_P+count_vote_F

                if (count == 3):
                    if ((count_vote_P/count)*100 >= 66):
                        self.__send_all(s,{'type':'msg', 'data':'11111111111'})
                        ui.setText("블록생성 성공")
                    else:
                        ui.setText("블록생성 실패")

                    count_vote_F = 0
                    count_vote_P = 0

def print_transaction(self,transaction,index):
    global ui
    if len(transaction)!=0 and index!=0:
        ui.setText2("====="+str(index)+"번          블록의          트랜잭션          시작
        =====\n")
        for i in range(len(transaction)):
            ui.setText2("트랜잭션번호 : "+str(transaction[i]['number'])+
                "\n발생시간 : "+str(transaction[i]['time'])+

```

```

        "\n출입자 : "+str(transaction[i]['user'])+
        "\n체크포인트 : "+str(transaction[i]['checkpoint'])+
        "\n개폐여부 : "+str(transaction[i]['pass_fail'])+"\n")
    ui.setText2("===== "+str(index)+"번          블록의          트랜잭션          끝
===== \n")
    else:
        ui.setText2("트랜잭션 : None\n\n")

def hi():
    global ui
    global block
    global checkpoint
    start_time = time.time()
    host = 'localhost'
    port = int(3000)
    block = Blockchain()

    ca = Node(host, port)
    ca.run_server()

checkpoint = ""

block = Blockchain()
block.new_block(1,1,1,1)
thrd = threading.Thread(target = hi, args={})
thrd.start()
app.exec_()

```

ica1.py

```

import time
from datetime import datetime
import json
import random
import hashlib
import threading
import pickle
import copy
from socket import *
from bc import Blockchain
import os

import sys
from PyQt5 import QtCore, QtGui, QtWidgets

f1=""

class Ui_Main_F(object):
    def setupUi(self, Main_F):
        Main_F.setObjectName("Main_F")
        Main_F.resize(969, 678)
        Main_F.setStyleSheet("background-color: rgb(255,254,239);")
        Main_F.setSizeGripEnabled(False)

```

```

self.label = QtWidgets.QLabel(Main_F)
self.label.setGeometry(QtCore.QRect(0, 0, 969, 51))
font = QtGui.QFont()
font.setFamily("맑은 고딕")
font.setPointSize(16)
font.setBold(True)
font.setWeight(75)
self.label.setFont(font)
self.label.setStyleSheet("background-color: rgb(170, 0, 0);")
self.label.setFrameShape(QtWidgets.QFrame.NoFrame)
self.label.setObjectName("label")
self.plainTextEdit = QtWidgets.QPlainTextEdit(Main_F)
self.plainTextEdit.setGeometry(QtCore.QRect(30, 130, 901, 501))
font = QtGui.QFont()
font.setFamily("맑은 고딕")
font.setPointSize(14)
self.plainTextEdit.setFont(font)
self.plainTextEdit.setFocusPolicy(QtCore.Qt.NoFocus)
self.plainTextEdit.setStyleSheet("background-color: rgb(255, 255, 255);\n"
"border: 3px solid #002060;")
self.plainTextEdit.setReadOnly(True)
self.plainTextEdit.setObjectName("plainTextEdit")

self.retranslateUi(Main_F)
QtCore.QMetaObject.connectSlotsByName(Main_F)

def retranslateUi(self, Main_F):
    _translate = QtCore.QCoreApplication.translate
    Main_F.setWindowTitle(_translate("Main_F", "GUI ICA1"))
    Main_F.setToolTip(_translate("Main_F",
"<html><head/><body><p>hi</p><p><br/></p></body></html>"))
    Main_F.setWhatsThis(_translate("Main_F",
"<html><head/><body><p>h</p></body></html>"))
    self.label.setText(_translate("Main_F",
align="center"><span
color:#ffffff;">Well-made</span></p></body></html>"))
    self.plainTextEdit.setPlainText(_translate("Main_F", "ICA1"))

def settext(self, str_):
    self.plainTextEdit.appendPlainText(str_)

app = QtWidgets.QApplication(sys.argv)
Main_F = QtWidgets.QDialog()
ui = Ui_Main_F()
ui.setupUi(Main_F)
Main_F.show()

class Node():
    def __init__(self, host, port):
        self.host = host
        self.port = int(port)
        self.nodes = []

    def __debug(self, msg):
        ui.settext('[%s] %s' % (str(threading.currentThread().getName()), msg))

```

```

def __debug_server(self, msg):
    ui.setText('[첫번째 중간인증기관 서버] %s' %(msg))

def __debug_client(self, msg):
    th_name = str(threading.currentThread().getName())
    if th_name == 'Thread-2':
        ui.setText('[첫번째 중간인증기관 서버] %s\n' %(msg))
    else:
        ui.setText('[첫번째 중간인증기관 서버] %s\n' %(msg))

def run_server(self):
    self.s = socket(AF_INET, SOCK_STREAM)
    self.s.bind((self.host, self.port))
    self.s.listen(5)

    self.__debug_server('서버 실행 (%d)\n' % self.port)
    recv_count = 0
    while True:
        self.__debug_server('연결대기 ... ')
        client_sock, client_addr = self.s.accept()
        self.nodes.append(client_sock)
        self.__debug_server('연결됨' + str(self.nodes[-1].getpeername()))

        t_recv = threading.Thread(target = self.recv_data, args={self.nodes[-1]})
        t_recv.start()
        recv_count+=1

def run_client(self, host, port):
    s = socket(AF_INET, SOCK_STREAM)
    s.connect((host, int(port)))
    ui.setText('[첫번째 인증기관 서버-출입문 연결] 연결됨'+str(s.getpeername()))
    return s

def __send(self, s, data):

    msg = pickle.dumps(data)
    s.sendall(msg)

def send_block(self, s):
    i = 4
    time.sleep(30)
    time.sleep(30)
    time.sleep(30)
    while True:
        blocks = block.new_block(block.hash_block(),2,3,i)
        self.__send(s, blocks)
        i += 1
        ui.setText('[첫번째 인증기관 서버] 생성된 블록을 중앙인증기관(CA)으로 전송\n')
        time.sleep(5)

def sign(self,s):
    time.sleep(10)
    self.__send(s, {'type':'verify' , 'cp_name':self.cp_name, 'cp_sign':self.cp_sign})

```

```

def recv_data(self, s):
    global ui
    ui.settext('[첫번째 중간인증기관 서버] 데이터 수신 대기중 ... ')
    while True:
        global block
        global ca_conn
        global names
        msg = s.recv(1024)
        data = pickle.loads(msg)

        if data:

            if(data.get('type') == 'transaction'):
                ui.settext('[출입문 수신] 트랜잭션 수신 완료')
                block.input_transaction(data)
                tranNsign = copy.deepcopy(data)
                tranNsign.update({'type':'tranNsign', 'cp_name':self.cp_name,
'cp_sign':self.cp_sign})

                ui.settext('[첫번째 인증기관 서버] 서명 검증을 위한 서명 전송.')
                ui.settext("\n"+str(tranNsign))

                self.__send(ca_conn, tranNsign)

            elif(data.get('type') == 'tranNsign'):
                data.update({'type':'transaction'})
                block.input_transaction(data)

            elif(data.get('type') == 'face'):
                name = data.get('data')
                n1 = name.replace("[('","")
                n2 = n1.replace("","")

                if(n2 in names):
                    passfail = data.get('passfail')
                    self.__send(s, {'type':'passfail', 'user':n2, 'passfail':passfail})

                else:
                    self.__send(s, {'type':'passfail', 'user':'unknown',
'passfail':"False"})

            elif(data.get('type') == 'new_name'):
                names.append(data.get('data'))

            elif(data.get('type') == 'sign_copy'):
                self.cp_name = data.get('cp_name')
                self.cp_sign = data.get('cp_sign')

            elif(data.get('type') == 'vote'):
                verify = data.get('data')

            if (verify==True):

```



```

        self.__send(s, {'type':'r_vote', 'data':1})
        ui.setText("생성된 블록에 대한 검증완료")
    else:
        ui.setText(verify)
        self.__send(s, {'type':'r_vote', 'data':0})

    elif(data.get('type') == 'block'):
        ui.setText("블록 수신 완료")

names=""
global ca_conn
def hi():
    global ui
    global block
    global checkpoint
    global names
    global ca_conn

    host = 'localhost'
    port = int(2000)
    block = Blockchain()
    block.new_block(1,1,1,1)
    com = 1

    ica = Node(host, port)

    ca_conn = ica.run_client(host, int(3000))
    ca_recv = threading.Thread(target = ica.recv_data, args={ca_conn})
    ca_recv.start()
    start_time = time.time()
    ca_vote = threading.Thread(target = ica.send_block, args={ca_conn})

    ca_vote.start()

    ica.run_server()

checkpoint = ""

block = Blockchain()
thrd = threading.Thread(target = hi, args={})
thrd.start()
app.exec_()

```

ica2.py

```

import time
from datetime import datetime
import json
import random
import hashlib
import threading
import pickle
import copy
from socket import *
from bc import Blockchain

```

```

import os

import sys
from PyQt5 import QtCore, QtGui, QtWidgets

f1=""

class Ui_Main_F(object):
    def setupUi(self, Main_F):
        Main_F.setObjectName("Main_F")
        Main_F.resize(969, 678)
        Main_F.setStyleSheet("background-color: rgb(255,254,239);")
        Main_F.setSizeGripEnabled(False)
        self.label = QtWidgets.QLabel(Main_F)
        self.label.setGeometry(QtCore.QRect(0, 0, 969, 51))
        font = QtGui.QFont()
        font.setFamily("맑은 고딕")
        font.setPointSize(14)
        font.setBold(True)
        font.setWeight(75)
        self.label.setFont(font)
        self.label.setStyleSheet("background-color: rgb(170, 0, 0);")
        self.label.setFrameShape(QtWidgets.QFrame.NoFrame)
        self.label.setObjectName("label")
        self.plainTextEdit = QtWidgets.QPlainTextEdit(Main_F)
        self.plainTextEdit.setGeometry(QtCore.QRect(30, 130, 901, 501))
        font = QtGui.QFont()
        font.setFamily("맑은 고딕")
        font.setPointSize(14)
        self.plainTextEdit.setFont(font)
        self.plainTextEdit.setFocusPolicy(QtCore.Qt.NoFocus)
        self.plainTextEdit.setStyleSheet("background-color: rgb(255, 255, 255);\n"
"border: 3px solid #002060;")
        self.plainTextEdit.setReadOnly(True)
        self.plainTextEdit.setObjectName("plainTextEdit")

        self.retranslateUi(Main_F)
        QtCore.QMetaObject.connectSlotsByName(Main_F)

    def retranslateUi(self, Main_F):
        _translate = QtCore.QCoreApplication.translate
        Main_F.setWindowTitle(_translate("Main_F", "GUI Test"))
        M a i n _ F . s e t T o o l T i p ( _ t r a n s l a t e ( " M a i n _ F " ,
"<html><head/><body><p>hi</p><p><br/></p></body></html>"))
        M a i n _ F . s e t W h a t s T h i s ( _ t r a n s l a t e ( " M a i n _ F " ,
"<html><head/><body><p>h</p></body></html>"))
        self.label.setText(_translate("Main_F",
"align=\"center\"><span style=\" color:#ffffff;\">중부대학교</span></p></body></html>"))
        self.plainTextEdit.setPlainText(_translate("Main_F", "ICA2"))

    def settext(self, str_):
        self.plainTextEdit.appendPlainText(str_)

app = QtWidgets.QApplication(sys.argv)
Main_F = QtWidgets.QDialog()
ui = Ui_Main_F()

```

```

ui.setupUi(Main_F)
Main_F.show()

class Node():
    def __init__(self, host, port):
        self.host = host
        self.port = int(port)
        self.nodes = []

    def __debug(self, msg):
        print('[%s] %s' % (str(threading.currentThread().getName()), msg))

    def __debug_server(self, msg):
        global ui
        ui.setText('[두번째 중간인증기관 서버] %s' %(msg))

    def __debug_client(self, msg):
        th_name = str(threading.currentThread().getName())
        if th_name == 'Thread-1':
            ui.setText('[두번째 중간인증기관(ICA2)-인증기관(CA) 연결] %s\n' %(msg))
        elif th_name == 'Thread-2':
            ui.setText('[인증기관(CA) 수신] %s\n' %(msg))
        else:
            ui.setText('[두번째 중간인증기관 서버] %s \n' %(msg))

    def run_server(self):
        self.s = socket(AF_INET, SOCK_STREAM)
        self.s.bind((self.host, self.port))
        self.s.listen(5)

        while True:
            client_sock, client_addr = self.s.accept()
            self.nodes.append(client_sock)
            self.__debug_server('연결됨' + str(self.nodes[-1].getpeername()))

            t_recv = threading.Thread(target = self.recv_data, args={self.nodes[-1]})
            t_recv.start()

    def run_client(self, host, port):
        s = socket(AF_INET, SOCK_STREAM)
        s.connect((host, int(port)))
        ui.setText('[두번째      중간인증기관      서버(ICA2)-인증기관(CA)]      연결됨'
'+str(s.getpeername())+'\n')
        return s

    def __send(self, s, data):
        msg = pickle.dumps(data)
        s.sendall(msg)

    def recv_data(self, s):
        global ui
        ui.setText('[두번째 중간인증기관 서버] 데이터 수신 대기중 ... \n')
        while True:
            msg = s.recv(1024)
            data = pickle.loads(msg)

```

```

        if data:

            if(data.get('type') == 'transaction'):
                ui.setText('[인증기관 수신(CA)] 트랜잭션 수신 완료')
                self.__send(s, {'type':'msg', 'data':'Receive tran'})
                self.__send(ca_conn, data)

            elif(data.get('type') == 'face'):
                if(name == data.get('data')):
                    self.__send(s, {'type':'passfail', 'user':name, 'passfail':1})
                else:
                    self.__send(s, {'type':'passfail', 'user':'unknown', 'passfail':0})

            elif(data.get('type') == 'sign'):
                pass

            elif(data.get('type') == 'sign_copy'):
                ui.setText('[인증기관 수신(CA)] 서명 수신 완료\n' +str(data)+"\n")
                self.cp_name = data.get('cp_name')
                self.cp_sign = data.get('cp_sign')

            elif(data.get('type') == 'vote'):
                verify = data.get('data')
                if (verify==True):
                    self.__send(s, {'type':'r_vote', 'data':1})
                    ui.setText('[두번째 인증기관 서버] 블록합의에 대한 동의 전송\n')
                else:
                    ui.setText(verify)
                    self.__send(s, {'type':'r_vote', 'data':0})

            elif(data.get('type') == 'block'):
                ui.setText("[두번째 인증기관 서버] 블록 수신 완료\n")

global ca_conn
def hi():
    global ui
    global block
    global checkpoint
    global names
    global ca_conn

    host = 'localhost'
    port = int(2002)
    block = Blockchain()
    block.new_block(1,1,1,1)
    com = 1

    ica = Node(host, port)

    ca_conn = ica.run_client(host, int(3000))
    ca_recv = threading.Thread(target = ica.recv_data, args={ca_conn})
    ca_recv.start()

```

```

start_time = time.time()

ica.run_server()

checkpoint = ""

block = Blockchain()
thrd = threading.Thread(target = hi, args={})
thrd.start()
app.exec_()

```

ica3.py

```

import time
from datetime import datetime
import json
import random
import hashlib
import threading
import pickle
import copy
from socket import *
from bc import Blockchain
import os

import sys
from PyQt5 import QtCore, QtGui, QtWidgets

f1=""

class Ui_Main_F(object):
    def setupUi(self, Main_F):
        Main_F.setObjectName("Main_F")
        Main_F.resize(969, 678)
        Main_F.setStyleSheet("background-color: rgb(255,254,239);")
        Main_F.setSizeGripEnabled(False)
        self.label = QtWidgets.QLabel(Main_F)
        self.label.setGeometry(QtCore.QRect(0, 0, 969, 51))
        font = QtGui.QFont()
        font.setFamily("맑은 고딕")
        font.setPointSize(14)
        font.setBold(True)
        font.setWeight(75)
        self.label.setFont(font)
        self.label.setStyleSheet("background-color: rgb(170, 0, 0);")
        self.label.setFrameShape(QtWidgets.QFrame.NoFrame)
        self.label.setObjectName("label")
        self.plainTextEdit = QtWidgets.QPlainTextEdit(Main_F)
        self.plainTextEdit.setGeometry(QtCore.QRect(30, 130, 901, 501))
        font = QtGui.QFont()
        font.setFamily("맑은 고딕")
        font.setPointSize(14)
        self.plainTextEdit.setFont(font)
        self.plainTextEdit.setFocusPolicy(QtCore.Qt.NoFocus)

```

```

        self.plainTextEdit.setStyleSheet("background-color: rgb(255, 255, 255);\n"
"border: 3px solid #002060;")
        self.plainTextEdit.setReadOnly(True)
        self.plainTextEdit.setObjectName("plainTextEdit")

        self.retranslateUi(Main_F)
        QtCore.QMetaObject.connectSlotsByName(Main_F)

    def retranslateUi(self, Main_F):
        _translate = QtCore.QCoreApplication.translate
        Main_F.setWindowTitle(_translate("Main_F", "GUI Test"))
        Main_F.setToolTip(_translate("Main_F",
"<html><head/><body><p>hi</p><p><br/></p></body></html>"))
        Main_F.setWhatsThis(_translate("Main_F",
"<html><head/><body><p>h</p></body></html>"))
        self.label.setText(_translate("Main_F",
align="center"><span style="color:#ffffff;">중부대학교</span></p></body></html>"))
        self.plainTextEdit.setPlainText(_translate("Main_F", "ICA3"))

    def settext(self, str_):
        self.plainTextEdit.appendPlainText(str_)

app = QtWidgets.QApplication(sys.argv)
Main_F = QtWidgets.QDialog()
ui = Ui_Main_F()
ui.setupUi(Main_F)
Main_F.show()

class Node():
    def __init__(self, host, port):
        self.host = host
        self.port = int(port)
        self.nodes = []

    def __debug(self, msg):
        print('[%s] %s' % (str(threading.currentThread().getName()), msg))

    def __debug_server(self, msg):
        global ui
        ui.settext('[세번째 중간인증기관 서버] %s' %(msg))

    def __debug_client(self, msg):
        th_name = str(threading.currentThread().getName())
        if th_name == 'Thread-1':
            ui.settext('[세번째 중간인증기관(ICA3)-인증기관(CA) 연결] %s\n' %(msg))
        elif th_name == 'Thread-2':
            ui.settext('[인증기관(CA) 수신] %s\n' %(msg))
        else:
            ui.settext('[세번째 중간인증기관 서버] %s\n' %(msg))

    def run_server(self):
        self.s = socket(AF_INET, SOCK_STREAM)
        self.s.bind((self.host, self.port))
        self.s.listen(5)

```

```

while True:
    client_sock, client_addr = self.s.accept()
    self.nodes.append(client_sock)
    self.__debug_server('연결됨' + str(self.nodes[-1].getpeername()))

    t_rcv = threading.Thread(target = self.rcv_data, args={self.nodes[-1]})
    t_rcv.start()

def run_client(self, host, port):
    s = socket(AF_INET, SOCK_STREAM)
    s.connect((host, int(port)))
    ui.settext('[세번째       중간인증기관       서버(ICA3)-인증기관(CA)]       연결됨'
'+str(s.getpeername())+"\n")
    return s

def __send(self, s, data):
    msg = pickle.dumps(data)
    s.sendall(msg)

def rcv_data(self, s):
    ui.settext('[세번째 중간인증기관 서버] 데이터 수신 대기중 ... \n')

    while True:
        msg = s.recv(1024)
        data = pickle.loads(msg)

        if data:
            if(data.get('type') == 'transaction'):
                ui.settext('[인증기관 수신(CA)] 트랜잭션 수신 완료')
                self.__send(s, {'type':'msg', 'data':'Receive tran'})
                self.__send(ca_conn, data)

            elif(data.get('type') == 'face'):
                if(name == data.get('data')):
                    self.__send(s, {'type':'passfail', 'user':name, 'passfail':1})
                else:
                    self.__send(s, {'type':'passfail', 'user':'unknown', 'passfail':0})

            elif(data.get('type') == 'sign'):
                pass

            elif(data.get('type') == 'sign_copy'):
                ui.settext('[인증기관 수신(CA)] 서명 수신 완료\n' +str(data)+"\n")
                self.cp_name = data.get('cp_name')
                self.cp_sign = data.get('cp_sign')

            elif(data.get('type') == 'vote'):
                verify = data.get('data')
                if (verify==True):
                    self.__send(s, {'type':'r_vote', 'data':1})
                    ui.settext('[세번째 인증기관 서버] 블록합의에 대한 동의 전송\n')
                else:
                    ui.settext(verify)

```

```

        self.__send(s, {'type':'r_vote', 'data':0})

    elif(data.get('type') == 'block'):
        i.settext("[세번째 인증기관 서버] 블록 수신 완료\n")

global ca_conn
def hi():
    global ui
    global block
    global checkpoint
    global names
    global ca_conn

    host = 'localhost'
    port = int(2003)
    block = Blockchain()
    block.new_block(1,1,1,1)
    com = 1

    ica = Node(host, port)

    ca_conn = ica.run_client(host, int(3000))
    ca_recv = threading.Thread(target = ica.recv_data, args={ca_conn})
    ca_recv.start()
    start_time = time.time()

    ica.run_server()

checkpoint = ""

block = Blockchain()
thrd = threading.Thread(target = hi, args={})
thrd.start()
app.exec_()

```

cp.py

```

from socket import *
import threading
import time
from datetime import datetime
import json
import random
import hashlib
from bc import Blockchain
import pickle
import serial

import sys
from PyQt5 import QtCore, QtGui, QtWidgets

f1=""

class Ui_Main_F(object):

```



```

def setupUi(self, Main_F):
    Main_F.setObjectName("Main_F")
    Main_F.resize(969, 678)
    Main_F.setStyleSheet("background-color: rgb(255,254,239);")
    Main_F.setSizeGripEnabled(False)
    self.label = QtWidgets.QLabel(Main_F)
    self.label.setGeometry(QtCore.QRect(0, 0, 969, 51))
    font = QtGui.QFont()
    font.setFamily("맑은 고딕")
    font.setPointSize(16)
    font.setBold(True)
    font.setWeight(75)
    self.label.setFont(font)
    self.label.setStyleSheet("background-color: rgb(170, 0, 0);")
    self.label.setFrameShape(QtWidgets.QFrame.NoFrame)
    self.label.setObjectName("label")
    self.plainTextEdit = QtWidgets.QPlainTextEdit(Main_F)
    self.plainTextEdit.setGeometry(QtCore.QRect(30, 130, 901, 501))
    font = QtGui.QFont()
    font.setFamily("맑은 고딕")
    font.setPointSize(14)
    self.plainTextEdit.setFont(font)
    self.plainTextEdit.setFocusPolicy(QtCore.Qt.NoFocus)
    self.plainTextEdit.setStyleSheet("background-color: rgb(255, 255, 255);\n"
"border: 3px solid #002060;")
    self.plainTextEdit.setReadOnly(True)
    self.plainTextEdit.setObjectName("plainTextEdit")

    self.retranslateUi(Main_F)
    QtCore.QMetaObject.connectSlotsByName(Main_F)

def retranslateUi(self, Main_F):
    _translate = QtCore.QCoreApplication.translate
    Main_F.setWindowTitle(_translate("Main_F", "GUI CP"))
    Main_F.setToolTip(_translate("Main_F",
"<html><head/><body><p>hi</p><p><br/></p></body></html>"))
    Main_F.setWhatsThis(_translate("Main_F",
"<html><head/><body><p>h</p></body></html>"))
    self.label.setText(_translate("Main_F",
align="center"><span style=" color:#ffffff;">Well-Made</span></p></body></html>"))
    self.plainTextEdit.setPlainText(_translate("Main_F", "Check Point"))

def settext(self, str_):
    self.plainTextEdit.appendPlainText(str_)

app = QtWidgets.QApplication(sys.argv)
Main_F = QtWidgets.QDialog()
ui = Ui_Main_F()
ui.setupUi(Main_F)
Main_F.show()

class Node():
    def __init__(self, host, port):
        self.host = host
        self.port = int(port)
        self.nodes = []

```

```

def __debug(self, msg):
    global ui
    ui.settext('[%s] %s' % (str(threading.currentThread().getName()), msg))

def __debug_server(self, msg):
    ui.settext('[출입문 서버] %s\n' % (msg))

def __debug_client(self, msg):
    th_name = str(threading.currentThread().getName())
    if th_name == 'MainThread':
        ui.settext('[출입문-첫번째 중간인증기관(ICA1) 연결] %s\n' % (msg))
    elif th_name == 'Thread-1':
        ui.settext('[첫번째 중간인증기관(ICA1) 수신] %s\n' % (msg))
    elif th_name == 'Thread-2':
        ui.settext('[출입문 발신] %s\n' % (msg))
    elif th_name == 'Thread-3':
        ui.settext('[출입문 발신] %s\n' % (msg))
    else:
        ui.settext('[출입문 서버] %s \n' % (msg))

def scanning(self, s):
    time.sleep(45)
    while True:
        self.__send(s, {'type': 'face', 'data': f1[0], 'passfail': f1[1], 'time': f1[2]})
        ui.settext('[출입문 서버] 생체 정보 전송\n')
        time.sleep(5)

def run_server(self):
    self.s = socket(AF_INET, SOCK_STREAM)
    self.s.bind((self.host, self.port))
    self.s.listen(5)

    self.__debug_server('서버 실행 (%d)\n' % self.port)
    recv_count = 0
    while True:
        self.__debug_server('연결대기 ... ')
        client_sock, client_addr = self.s.accept()
        self.nodes.append(client_sock)
        self.__debug_server('연결됨' + str(self.nodes[-1].getpeername()))

        r_recv = threading.Thread(target = self.recv, args={self.nodes[-1]})
        r_recv.start()

def run_client(self, host, port):
    s = socket(AF_INET, SOCK_STREAM)
    s.connect((host, int(port)))
    ui.settext('[출입문-첫번째 중간인증기관(ICA1) 연결]' + str(s.getpeername()) + "\n")
    return s

def __send(self, s, data):
    msg = pickle.dumps(data)
    s.sendall(msg)

def recv(self, s):

```

```

ui.setText("[출입문 서버] 데이터 수신 대기중 ... \n")
global block
global times
global checkpoint
times = 0
tran_data = block.current_transactions[-1]
while True:

    msg = s.recv(1024)
    data = pickle.loads(msg)
    if data:
        if(data.get('type') == 'passfail'):
            time_ = times
            block.new_transaction(tran_data['number']+1,      data.get('user'),
checkpoint, data.get('time'), data.get('passfail'))

            times = data.get('time')
            if (data.get('passfail')=='True'):
                arduino_sign = 10
                arduino.write(arduino_sign)
                ui.setText("[출입문 서버] 출입 허가\n")
            elif(data.get('passfail')=='False'):
                arduino_sign = 100
                arduino.write(arduino_sign)
                ui.setText("[출입문 서버] 출입 불허. 사용자를 등록해주세요.\n")
            if (times ==time_):
                ui.setText("[출입문 서버] 동일한 시간대의 출입기록. 대기중 ... \n")
            else:
                tran_data = block.current_transactions[-1]
                self.__send(s, tran_data)
                ui.setText("[출입문 서버] 트랜잭션 생성완료. 중간 인증서버로 전송
(ICA1) \n")

        if(data.get('type') == 'face'):
            f = str(data['data'])
            global f1
            f1 = f.split(",")
            self.__send(s, {'type' : 'r_face', 'data' : "전송완료"})

        if(data.get('type')=='new_face'):
            self.__send(cp_s, {'type':'new_name', 'data':data.get('data')})
            ui.setText("[출입문 서버] 새로운 사용자 정보 전송\n")

def hi():
    global ui
    global block
    global checkpoint
    global arduino

    arduino = serial.Serial('COM4', 9600)
    checkpoint = 'checkpoint1'
    host = 'localhost'
    port = int(1234)

    block_data = block.new_block(1,1,1,1)
    create_tran = block.new_transaction(1, 'user1', 'checkpoint1', 1, 1)

```

```

tran_data = block.current_transactions[-1]

cp = Node(host, port)
cp_s = cp.run_client('localhost', 2000)

t_recv = threading.Thread(target = cp.recv, args={cp_s})
t_recv.start()
t_scanning = threading.Thread(target = cp.scanning, args={cp_s})
t_scanning.start()

cp.run_server()

checkpoint = ""

block = Blockchain()
thrd = threading.Thread(target = hi, args={})
thrd.start()
app.exec_()

```

face_recog.py

```

import face_recognition
import cv2
import numpy as np
import os
import sys
import time
import logging
from datetime import datetime
from watchdog.observers import Observer
from watchdog.events import FileSystemEventHandler
from socket import *
import threading
import time
import json
import random
import hashlib
import pickle
import sys
from PyQt5 import QtCore, QtGui, QtWidgets

video_capture = cv2.VideoCapture(1)

class Ui_Main_F(object):
    def setupUi(self, Main_F):
        Main_F.setObjectName("Main_F")
        Main_F.resize(969, 678)
        Main_F.setStyleSheet("background-color: rgb(255,254,239);")
        Main_F.setSizeGripEnabled(False)
        self.label = QtWidgets.QLabel(Main_F)
        self.label.setGeometry(QtCore.QRect(0, 0, 969, 51))
        font = QtGui.QFont()
        font.setFamily("맑은 고딕")

```

```

font.setPointSize(16)
font.setBold(True)
font.setWeight(75)
self.label.setFont(font)
self.label.setStyleSheet("background-color: rgb(170, 0, 0);")
self.label.setFrameShape(QtWidgets.QFrame.NoFrame)
self.label.setObjectName("label")
self.plainTextEdit = QtWidgets.QPlainTextEdit(Main_F)
self.plainTextEdit.setGeometry(QtCore.QRect(30, 130, 901, 501))
font = QtGui.QFont()
font.setFamily("맑은 고딕")
font.setPointSize(14)
self.plainTextEdit.setFont(font)
self.plainTextEdit.setFocusPolicy(QtCore.Qt.NoFocus)
self.plainTextEdit.setStyleSheet("background-color: rgb(255, 255, 255);\n"
"border: 3px solid #002060;")
self.plainTextEdit.setReadOnly(True)
self.plainTextEdit.setObjectName("plainTextEdit")

self.retranslateUi(Main_F)
QtCore.QMetaObject.connectSlotsByName(Main_F)

def retranslateUi(self, Main_F):
    _translate = QtCore.QCoreApplication.translate
    Main_F.setWindowTitle(_translate("Main_F", "GUI FACE"))
    Main_F.setToolTip(_translate("Main_F",
"<html><head/><body><p>hi</p><p><br/></p></body></html>"))
    Main_F.setWhatsThis(_translate("Main_F",
"<html><head/><body><p>h</p></body></html>"))
    self.label.setText(_translate("Main_F",
align="center"><span
color:#ffffff;">Well-made</span></p></body></html>"))
    self.plainTextEdit.setPlainText(_translate("Main_F", "Face"))

def settext(self, str_):
    self.plainTextEdit.appendPlainText(str_)

app = QtWidgets.QApplication(sys.argv)
Main_F = QtWidgets.QDialog()
ui = Ui_Main_F()
ui.setupUi(Main_F)
Main_F.show()

class Node():
    def __init__(self, host, port):
        self.host = host
        self.port = int(port)
        self.nodes = []

    def __debug(self, msg):
        ui.setText(['%s' %s' % (str(threading.currentThread().getName()), msg))

    def __debug_server(self, msg):
        ui.setText(['얼굴인식서버] %s\n' %(msg))

    def __debug_client(self, msg):

```

```

th_name = str(threading.currentThread().getName())
if th_name == 'MainThread':
    ui.setText('[출입문-얼굴인식서버 연결] %s\n' %(msg))
else:
    ui.setText('[얼굴인식서버] %s\n' %(msg))

def scanning(self, s):
    self.__send(s, {'type':'face', 'data':result_value})
    self.__debug_client('생체정보 전송완료\n')
    time.sleep(1)

def new_user(self, s):
    self.__send(s, {'type':'new_face', 'data':new_name})
    self.__debug_client('새로운 사용자 정보 전송완료 \n')
    time.sleep(1)

def run_server(self):
    self.s = socket(AF_INET, SOCK_STREAM)
    self.s.bind((self.host, self.port))
    self.s.listen(5)

    self.__debug_server('서버 실행 (%d)\n' % self.port)

    while True:
        self.__debug_server('연결대기 ... ')
        client_sock, client_addr = self.s.accept()
        self.nodes.append(client_sock)
        self.__debug_server('연결됨' + str(self.nodes[-1].getpeername()))

        t_recv = threading.Thread(target = self.__recv, args={self.nodes[-1]})
        t_recv.start()

def run_client(self, host, port):
    s = socket(AF_INET, SOCK_STREAM)
    s.connect((host, int(port)))
    self.__debug_client('연결됨 ' + str(s.getpeername()))
    return s

def __send(self, s, data):
    msg = pickle.dumps(data)
    s.sendall(msg)

def recv(self, s):
    self.__debug_client('연결완료. 데이터 수신 대기중 ... \n')
    while True:
        msg = s.recv(1024)
        data = pickle.loads(msg)
        if data:
            self.__debug_client('받은정보 : ' + str(data))
            if(data.get('type') == 'r_face'):
                ui.setText(str(data))

class Handler(FileSystemEventHandler):
    def on_created(self, event):
        global new_name

```

```

        now = datetime.now()
        ui.settext("\n      training에      새로운      사용자가      추가되었습니다.", "[%s-%s-%s-%s:%s:%s]\n'
                    % (now.year, now.month, now.day, now.hour, now.minute, now.second))
        images_name = os.listdir(path_dir)
        names = "".join(map(str, images_name)).split(".jpg")
        del names[len(names)-1]

        for new_name in names :
            if new_name not in known_face_names:
                known_face_names.append(new_name)
                image_load = face_recognition.load_image_file(path_dir + "\\\" +
new_name + ".jpg")
                encoding = face_recognition.face_encodings(image_load)[0]
                known_face_encodings.append(encoding)
                new_send = threading.Thread(target = face.new_user, args={face_c})
                new_send.start()

path_dir = r'C:\Users\lee\Desktop\training'
images_name = os.listdir(path_dir)
names = "".join(map(str, images_name)).split(".jpg")
del names[len(names)-1]

known_face_encodings = []
known_face_names = []

for image in images_name:
    image_load = face_recognition.load_image_file(path_dir + "\\\" + image)
    face_encoding = face_recognition.face_encodings(image_load)[0]
    known_face_encodings.append(face_encoding)
    known_face_names.append((image.split(".")[0])

class Face():
    global result_value
    global face_c
    global face

    face_locations = []
    face_encodings = []
    face_names = []
    process_this_frame = True

    while True:
        ret, frame = video_capture.read()
        small_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)
        rgb_small_frame = small_frame[:, :, :-1]

        if process_this_frame:
            face_locations = face_recognition.face_locations(rgb_small_frame)
            face_encodings = face_recognition.face_encodings(rgb_small_frame,
face_locations)
            face_names = []

```

```

        for face_encoding in face_encodings:
            face_distances = face_recognition.face_distance(known_face_encodings, face_encoding)
            min_value = min(face_distances)
            name = "Unknown"
            now = datetime.now()

            if min_value < 0.435:
                index = np.argmin(face_distances)
                name = known_face_names[index]
                value = "True"
                ui.setText("등록된 사용자 입니다. [True]")
            else :
                value = "False"
                ui.setText("등록되지 않은 사용자 입니다. [False]")
            day = (name,value,
                '%s-%s-%s-%s:%s:%s' % (now.year, now.month, now.day,
now.hour, now.minute, now.second))
            result_value=[day]
            face_names.append(name)

            ui.setText("Face Recognition")
            host = 'localhost'
            port = int(2345)
            face =Node(host,port)
            face_c = face.run_client('localhost', 1234)
            f_rcv = threading.Thread(target =face.rcv, args = {face_c})
            f_rcv.start()
            f_scanning = threading.Thread(target=face.scanning, args = {face_c})
            f_scanning.start()

process_this_frame = not process_this_frame

event_handler = Handler()
observer = Observer()
observer.schedule(event_handler, path_dir, recursive=True)
observer.start()
time.sleep(1)
observer.stop()

for (top, right, bottom, left), name in zip(face_locations, face_names):
    top *= 4
    right *= 4
    bottom *= 4
    left *= 4

    cv2.rectangle(frame, (left, top), (right, bottom), (0, 0, 255), 2)
    cv2.rectangle(frame, (left, bottom - 35), (right, bottom), (0, 0, 255),
cv2.FILLED)
    font = cv2.FONT_HERSHEY_DUPLEX
    cv2.putText(frame, name, (left + 6, bottom - 6), font, 1.0, (255, 255, 255),
1)

    cv2.imshow('Video', frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):

```



```
        break

    video_capture.release()
    cv2.destroyAllWindows()

thrd = threading.Thread(target = Face, args={})
thrd.start()
app.exec_()
```

블록체인 기반 출입통제 시스템

2019. 10. 29

중부대학교 정보보호학과

지도교수 : 양환석 교수님



2 조

조영선
김민정
김예진
이정한
조수홍

목 차



- 조원 편성
- 주제 선정
- 구 상 도
- 추진 경과
- 개발 환경 및 개발 내용
- 개발 시스템 운영
- 결론 및 기대효과

조원 편성



이름	역할
조영선(조장)	얼굴인식기술 개발, 작품 총괄
김민정	블록체인 개발, ppt 및 보고서 작성
김예진	블록체인 개발, GUI 개발
이정환	블록체인 개발, GUI 개발
조수홍	블록체인 개발, ppt 및 보고서 작성

3

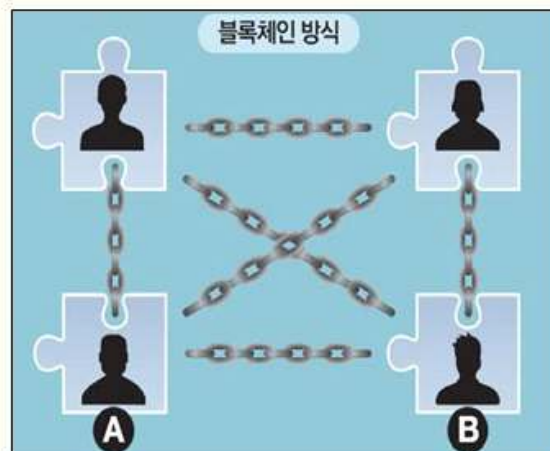
주제 선정(1/2)



블록체인이란?

‘블록’으로 불리는 데이터들이 연결된 체인 형태의 분산 원장 기술

- ◆ 중앙집중기관 없이 시스템 참가자들이 공동으로 거래 정보를 기록·보관·검증
- ▷ 거래 정보의 신뢰성 확보
- ▷ 데이터 위·변조 방지
- ▷ 탈중앙화, 보안성, 투명성을 보장



4

주제 선정(2/2)



주제 선정

[구축사례] **정부청사 출입통제 얼굴인식 시스템**

얼굴인식 솔루션 도입해 출입보안 강화

[컴퓨터
설비
수행
을 초
이제
가 되
사에
운전면허증, 스마트폰에 담긴다.. **개인정보는 블록체인
기술로 보호**

이예 : 도예리 기자 | 2019-10-02 14:40:48



스마트폰에 운전면허증이 담긴다. 운전면허증 상 개인정보는 블록체인 기술로 보호된다.

2일 경찰청은 서울 중구 롯데호텔에서 '모바일 운전면허 확인 서비스'를 추진하기 위해 SK텔레콤, KT, LG유플러스 등 통신 3사와

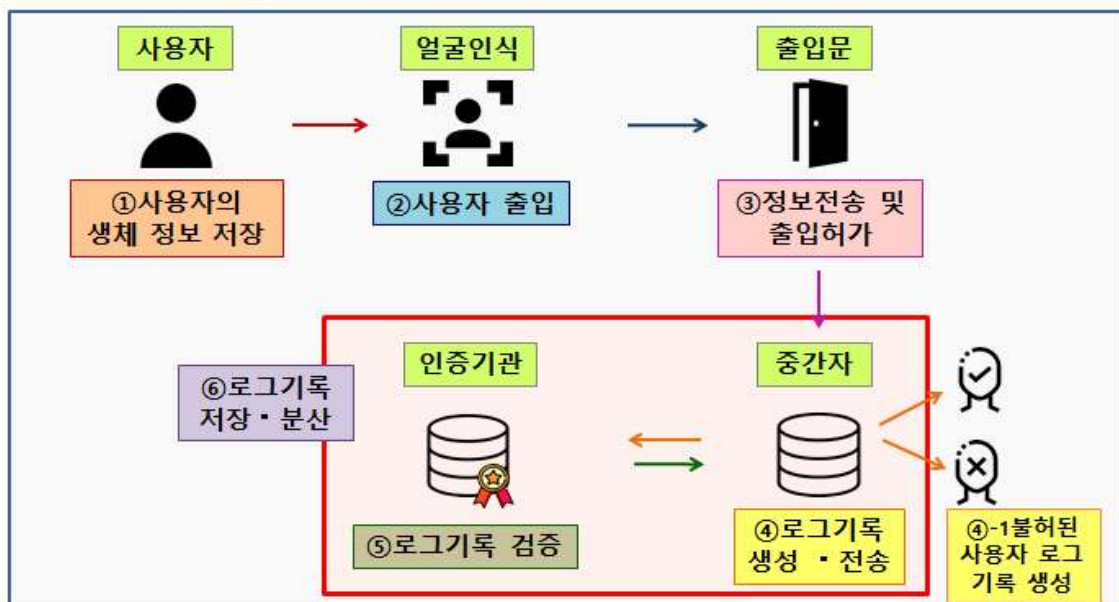
**출입기록의 데이터 위·변조에 대한 위험성을 줄이기 위해
블록체인 기술을 사용하여 출입통제 시스템 구현**

5

구상도



동작 원리



6

추진 경과



추진기간 (2019년) 수행 업무	3월	4월	5월	6월	7월	8월	9월	10월
자료 조사 및 연구								
얼굴인식 기술개발								
블록체인 개발								
GUI 개발								
시스템 검증 및 보완								

7

개발 환경 및 개발 내용(1/8)



개발 환경

Tools

- 얼굴인식 : Dlib , opencv
- 블록체인 : hashlib, socket

OS

Windows 10

Development Language

Python 3.6

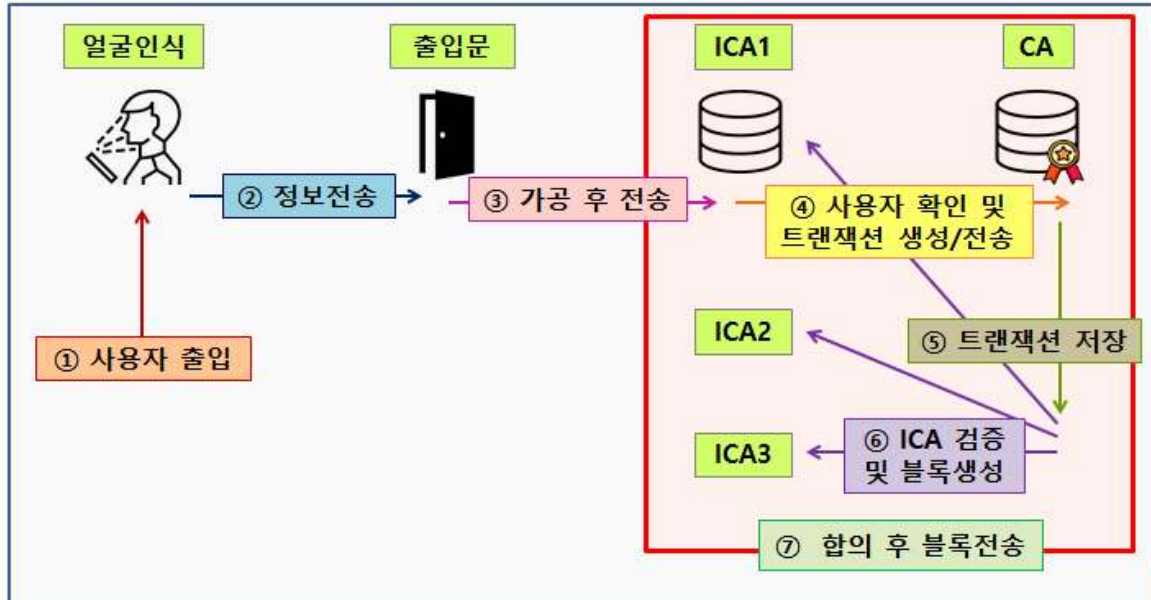
8

개발 환경 및 개발 내용(2/8)



블록체인 활용체계

Certificate Authority ⇨ 인증기관
Intermediate Certificate Authority ⇨ 중간 인증기관



9

개발 환경 및 개발 내용(3/8)



얼굴인식 생체정보 관리(1/2)

```
video_capture = cv2.VideoCapture(1)

path_dir = r'C:\Users\whdud\Desktop\training'
images_name = os.listdir(path_dir)

names = ""
known_face_encodings = []
known_face_names = []

for image in images_name:
    print((image.split(".")[0]))
    name_image = face_recognition.load_image_file(path_dir + "\\" + image)
    name_face_encoding = face_recognition.face_encodings(name_image)[0]
    known_face_encodings.append(name_face_encoding)
    known_face_names.append((image.split(".")[0]))
```

생체정보 저장
- 캠 활성화

생체정보 저장
- 인식과정

① 캠 활성화 ② training 파일의 저장된 사용자 정보를 가공 후 저장
(사용자 생체정보 생성 · 저장)

10

개발 환경 및 개발 내용(4/8)



얼굴인식 생체정보 관리(2/2)

```
if process_this_frame:
    face_locations = face_recognition.face_locations(rgb_small_frame)
    face_encodings = face_recognition.face_encodings(rgb_small_frame, face_locations)
    face_names = []

    for face_encoding in face_encodings:
        face_distances = face_recognition.face_distance(known_face_encodings, face_encoding)
        min_value = min(face_distances)
        name = "Unknown"
        now = datetime.now()

        if min_value < 0.435:
            index = np.argmin(face_distances)
            name = known_face_names[index]
            value = "True"
            print("등록된 사용자 입니다. [True]")
        else:
            value = "False"
            print("등록되지 않은 사용자 입니다. [False]")
        day = (name, value,
              '%s-%s-%s-%s-%s-%s' % (now.year, now.month, now.day, now.hour, now.minute, now.second))
        result_value=[day]
```

생체정보 관리 · 전송

현재 비디오 프레임 속 얼굴과
저장된 생체정보가 일치하는지 확인 후 결과 전송

11

개발 환경 및 개발 내용(5/8)



트랜잭션 생성 및 전송

```
elif(data.get('type') == 'face'):
    name = data.get('data')
    n1 = name.replace("(", "")
    n2 = n1.replace(")", "")

    if(n2 in names):
        # name = data.get('name')
        if data:
            if(data.get('type') == 'transaction'):
                block.input_transaction(data)
                tranNsign = copy.deepcopy(data)
                tranNsign.update({'type': 'tranNsign', 'cp_name': self.cp_name, 'cp_sign': self.cp_sign})
                global ui
                ui.setText("#n"+str(tranNsign))
                self.__debug_client("서명 검증을 위한 서명 전송.")

                self.__send(ca_conn, tranNsign)

            elif(data.get('type') == 'tranNsign'):
                data.update({'type': 'transaction'})
                block.input_transaction(data)
```

사용자 확인 과정

트랜잭션 생성 · 전송

사용자 확인 절차 및 트랜잭션 생성 · 전송

12

개발 환경 및 개발 내용(6/8)



블록 생성 · 전송 및 합의(1/2)

```
class Blockchain(object):
    def __init__(self):
        self.chain = []
        self.current_transactions = []

        self.vote = []
        self.last_block = {}
        self.new_block(previous_hash = 1)
```

블록 생성

```
def new_block(self, previous_hash, proof, timestamp, difficulty):
    def new_transaction(self, number, user, checkpoint, time):
```

트랜잭션 생성

```
self.current_transactions.append({
    'type': 'transaction',
    'number': len(self.current_transactions)+1,
    'user': user,
    'checkpoint': checkpoint,
    'time': time,
    'pass_fail': pass_fail,
})
```

```
return self.last_block['index'] + 1
```

트랜잭션 생성과 블록 생성

13

개발 환경 및 개발 내용(7/8)



블록 생성 · 전송 및 합의(2/2)

```
def verify_sign(self, vk_sign, vk_name):
    self.vk_sign = vk_sign[:48]
    global ui
```

검증 과정

```
with open('verify_signfile.pickle', mode='rb') as file:
```

```
if data:
    if(data.get('type') == 'tranNsign'):
        sign_PF = self.verify_sign(data.get('cp_sign'), data.get('cp_name'))
        if sign_PF:
            del data['cp_name']
            del data['cp_sign']
            ui.settext(str(data))
            self.__send_all(s, data)
```

생성된 블록에 대한 합의과정

```
elif(data.get('type') == 'block'):
    self.__send_all(s, {'type': 'vote', 'data': sign_PF}, 1)

elif(data.get('type') == 'r_vote'):
    if (data.get('data')==0):
        count_vote_F +=1
```

- ① 신뢰할 수 있는 정보(트랜잭션) 인지에 대한 검증
- ② ICA간 합의 진행 ③ 블록전송

14

개발 환경 및 개발 내용(8/8)



GUI

```
def retranslateUi(self, Main_F):
    _translate = QtCore.QCoreApplication.translate
    Main_F.setWindowTitle(_translate("Main_F", "GUI Test"))
    Main_F.setToolTip(_translate("Main_F", "<html><head/><body><p>hi</p><p><br/></p></body></html>"))
    Main_F.setWhatsThis(_translate("Main_F", "<html><head/><body><p>hi</p></body></html>"))
    self.label.setText(_translate("Main_F", "<html><head/><body><p align='center'><span style='<div data-bbox="774 204 819 220" data-label="Text">


GUI


```

시스템의 전반적인 그래픽 사용자 인터페이스

15

개발 시스템 운영 (1/6)



시스템 운영(1/6) - 통신관리



16

개발 시스템 운영 (2/6)



시스템 운영(2/6) - 사용자관리

사용자 관리 | 로그 기록 | 실시간 블록

등록된 사용자 수 : 5

메인 화면

사용자 추가·삭제 버튼

사용자 삭제완료

사용자 추가 | 사용자 추가화면

사용자 정보

<input type="checkbox"/>	사용자 이름	전화번호	해당부서
<input type="checkbox"/>	김민정	010-1111-1111	개발
<input type="checkbox"/>	김예진	010-2222-2222	개발
<input type="checkbox"/>	조영선	010-5555-5555	개발
<input type="checkbox"/>	조수룡	010-6666-6666	개발

사용자 이름
김민정

이메일 주소
joognbu1@naver.com

전화번호
010-1111-1111

해당부서
개발

17

개발 시스템 운영 (3/6)



시스템 운영(3/6) - 출입관리

GUI FACE

얼굴인식 서버-사용자 판별

중부대학교

출입문 서버-출입관리

중부대학교

출입불허

Checkpoint

[출입문 서버] **출입불허** 사용자를 등록해주세요

[출입문 서버] 트랜잭션 생성완료. 중간 인증서

[출입문 서버] 생체정보 전송완료

출입 불허

18

개발 시스템 운영 (4/6)



시스템 운영(4/6) - 출입관리

GUI FACE

중부대학

얼굴인식 서버-사용자 판별

출입문 서버-출입관리

중부대학교

Checkpoint

[출입문 서버] **출입허가**

[출입문 서버] 트랜잭션 생성완료. 중간 인증서

[출입문 서버] 생체정보 전송완료

출입허가

19

개발 시스템 운영 (5/6)



시스템 운영(5/6) - 출입관리

사용자 관리 로그 기록 실시간 불록 **출입기록 관리-마지막 출입시간 변경**

등록된 사용자 수 : 4 추가 삭제

<input type="checkbox"/>	사용자 이름	전화번호	해당부서	이메일주소	마지막 출입시간
<input type="checkbox"/>	김민정	010-1111-1111	개발	joongbu1@naver.com	2019년 10월 09일
<input type="checkbox"/>	김예진	010-2222-2222	개발	joongbu2@naver.com	2019년 10월 09일
<input type="checkbox"/>	조영선	010-5555-5555	개발	joongbu4@naver.com	2019년 10월 11일
<input type="checkbox"/>	조수룡	010-6666-6666	개발	joongbu5@naver.com	2019년 10월 09일
<input type="checkbox"/>	조영선	010-5555-5555	개발	joongbu4@naver.com	2019년 10월 09일
<input type="checkbox"/>	조영선	010-5555-5555	개발	joongbu4@naver.com	2019년 10월 11일

< 1 >

출입한 사용자의 정보에 따라 출입시간 변경

20

개발 시스템 운영 (6/6)



시스템 운영(6/6) - 블록관리

GUI Test

중부대학교

중간 인증기관 서버
-트랜잭션 - 블록 전송

ICA1
[첫 번째
[첫 번째
[첫 번째

사용자 관리
로그 기록
실시간 블록
메인화면-생성된 블록 확인

블록번호 : 4
발생시간 : 2019.10.07.16:52
이전해쉬 : f665a62063fa892e4ad3236ea3c3419f6f09f9ddfa6a38991727c49bf87cc045
=====4번 블록의 트랜잭션 시작=====

트랜잭션번호 : 1
발생시간 : None
출입자 : unknown
체크포인트 : checkpoint1
개폐여부 : False
=====4번 블록의 트랜잭션 끝=====

실시간 블록 화면에서 생성된 블록 확인

21

결론 및 기대 효과



○ 결 론

- 자체 기술력으로 얼굴인식 기술을 구현하고, 이를 기반으로 출입통제 및 블록체인을 개발하는데 성공
- 조원들에게 적절히 임무를 분담하여 필요 기술을 직접 구현하고 팀워크로 연구하는 조직체제를 가동, 기술역량을 배가

○ 기대효과

- 블록체인 기반 출입통제 시스템을 완성함으로써 데이터 위·변조를 방지하고 얼굴인식 기술 등을 응용, 출입통제 체제의 보안성과 편의성을 향상
- 인증된 사용자만을 출입통제함으로써 비인증 외부인의 출입을 차단, 외적 위험요소를 1차적으로 제거. 끝.

22

Q & A

감사합니다

23