블록체인을 활용한 새로운 보험대리점 수수료 지급 방식 구현

박재훈*, 강예준**, 김원웅**, 서화정*†
*한성대학교 IT 융합공학부 (대학원생)
**한성대학교 IT 융합공학부 (학부생)
*†한성대학교 IT 융합공학부 (교수)
p9595jh@gmail.com, etus1211@gmail.com, dnjsdndeee@gmail.com, hwajeong84@gmail.com

Implementation of New Method of GA's Fee Payment using Blockchain

Jae-Hoon Park*, Yae-Jun Kang**, Won-Woong Kim*, Hwa-Jeong Seo*†
*Division of IT Convergence Engineering, Hansung University (Graduate student)
**Division of IT Convergence Engineering, Hansung University (Undergraduate student)
*†Division of IT Convergence Engineering, Hansung University (Professor)

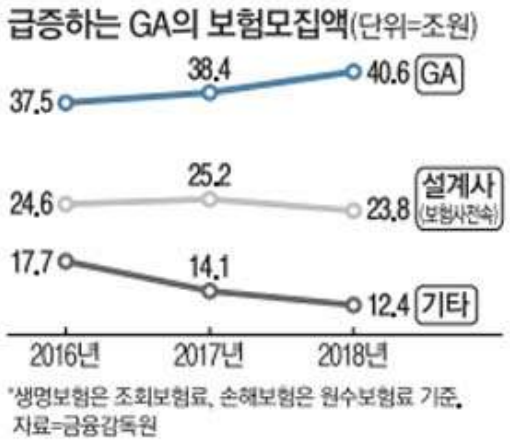
요 약

예로부터 보험은 보험사가 전담하여 판매하였었다. 보험사에 설계된 보험사가 오직 그 보험사만의 보험 상품을 판매하는 형태로 판매가 진행되었었다. 하지만 현재는 2005 년 보험대리점이 생겨나고서 보험대리점에서 다양한 보험사들의 보험 상품들을 판매하면서 보험사로부터 수수료를 받는 새로운 형태의 판매가 진행되고 있다. 그런데 이 수수료를 지급하는 방식에 대해, 지급 내역을 보험사에서 보험대리점으로 메일로 전송하는 식으로 보안성이 떨어지는 방식을 취하고 있으며 데이터의 형식도 모두 달라 보험대리점에서 따로 취합을 해줘야 하는 일이 발생한다. 본 논문에서는 이것을 블록체인을 통해 신뢰성 높은 방식의 데이터 전송을 지원하며 전송 형식 또한 규격화 하면서 보험대리점에서의 부담 또한 덜어주는 시스템을 구현하였다. 이 시스템을 통해 보안 및 업무의 낭비가 줄어드는 것을 기대한다.

1. 서론

보험 판매는 전통적으로 보험사에 직접 소속된 설계사들이 도맡아 판매 하였었다. 하지만 2005 년 외국계 보험회사 전문가들로 구성된 보험대리점(GA, General Agency)가 생겨난 이후로 보험대리점 소속 설계사들이 보험사로부터 보험을 얻어와 그것을 판매하기 시작하였다. 보험대리점은 특정 보험사에 종속되어 있지 않기에 보험대리점 소속 설계사들은 보험사의 종류에 관계 없이 다양한 종류의 보험을 판매할 수 있다. 또한 보험사는 이렇게 보험이 판매될 경우 해당 보험대리점에 판매한 것에 대한 수수료를 지급하게 된다.

보험대리점은 나날이 커져가고 있으며, 대형 GA 또한 많이 등장하고 있다. GA 를 통한 보험 계약은 다양한 보험사의 보험을 비교하여 가입할 수 있다는 장점이 있으며, 이러한 장점을 바탕으로 GA 시장 또한 커지고 있다. 실제로 보험대리점 소속 설계사의 수는 이미 보험사 전속 설계사의 수를 뛰어넘었다.



(그림 1) GA 의 보험 모집액[1]

이렇듯 GA 가 크게 성장하였으나, 보험사에서 GA 에 수수료를 지급하는 방식은 각 보험사마다 모두 상이하다. 각 보험사들이 엑셀을 통해 각자의 양식을

통해 수수료에 대한 정보를 보내고 있으며, GA 에서는 이러한 각기 다른 양식을 자신의 양식에 맞추어 새로 취합하는 과정이 필요하다. 또한, 단순히 메일을 통해 보내기 때문에 보험사와 보험대리점 간의 데이터가 무결성이 유지되지 않을 가능성 또한 존재한다.

본 논문에서는 블록체인을 활용하여 보험사에서 보험대리점에 수수료를 지급하는 양식 및 지급 프로토콜 자체에 대해 새로운 방식을 적용한 시스템을 구현하였다.

2. 관련 연구

2.1 블록체인

블록체인은 데이터를 블록 형태로 만든 뒤 여러 블록을 하나의 체인 형태로 엮어, 각 데이터의 무결성을 보장하고자 하는 Peer-to-Peer(P2P) 네트워크 방식의 일종이다. 최초의 블록체인은 사토시 나카모토가 비트코인을 개발할 때 이용한 것이다[2]. 암호화폐의 특성상 중앙이 따로 없는데, 이러한 '중앙이 존재하지 않는' 분산원장 방식은 비잔틴 장군 문제라는 데이터 위/변조의 확인이 힘든 방식에 대해 취약하였다[3]. 즉, P2P 네트워크 상에서 누군가가 데이터를 악의적으로 변조하더라도, 아니면 시스템에 의해 데이터의 일부가 손상되더라도 그것에 대한 확인이 어려웠었다. 하지만 비트코인에서는 각 블록이 자신의 앞/뒤 블록의 해시값을 가짐으로써 무결성에 대한 보장이 가능해졌다.

블록체인은 합의 프로토콜이라고 불리는 과정을 통해 새로운 블록을 생성할 수 있다. 네트워크 참여자들의 특정 퍼센테이지가 새로운 블록을 생성하는 것에 동의하게 되면 블록이 생성될 수 있는 것이다. 비트코인은 '작업증명'이라는, 컴퓨팅 파워를 이용하여 새로운 블록의 해시를 맞추는 방식의 합의 프로토콜을 이용한다. 작업증명 이외에도 지분증명, 위임지분증명 등의 다양한 종류의 합의 프로토콜이 존재한다.

2.2. 하이퍼레저 패브릭

본 논문에서는 프라이빗 블록체인 플랫폼의 일종인 하이퍼레저 패브릭을 이용하여 시스템을 구현하였다. 하이퍼레저 패브릭은 리눅스 재단에서 진행 중인 하이퍼레저 프로젝트로부터 나온 프레임워크로, IBM 에서 개발을 담당하고 있다. CA(Certificate Authority)를 이용하여 네트워크 참여자에 대한 인증을 거칠 수 있으며, '체인코드'라는 스마트 계약을 Go, node.js, Java 와 같은 일반 프로그래밍 언어를 통해 호스팅하여 비즈니스 구현에 적합하다[4]. 본 논문에서 이용한 하이퍼레저 패브릭은 2.2.2 LTS 버전으로, 합의 프로토콜은 RAFT 를 사용하고 있다. RAFT 는 대량의 데이터 처리에 더 적합한 방식이며 대표자가 참여자들의 동의를 얻어 확인이 끝나면 새로운 블록이 생성되는 방식을 취하고 있다.

하이퍼레저 패브릭의 구조는 크게 하나의 네트워크 내에 여러 채널이 있고, 각 채널을 조직들이 이루는 형태로 되어 있다. 각 조직은 피어를 가지고 있으며 체인코드는 피어에 설치되어 돌아가게 된다.

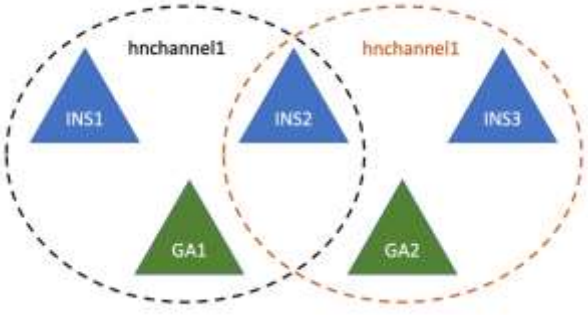
2.3. 도커

하이퍼레저 패브릭은 도커(docker)라는 컨테이너 시스템을 이용하여 실행된다. 도커는 실행환경이나 프로그램을 '이미지'로 만든 뒤, 그것을 '컨테이너'라는 하나의 추상화 단위로써 실행시킨 뒤 관리한다. 하이퍼레저 패브릭에서는 피어, 데이터베이스, CA 조직 등이 하나의 서비스로서 도커를 통해 구동되게 된다.

3. 보험대리점 수수료 지급 시스템

본 논문에서는 하이퍼레저 패브릭을 이용하여 보험대리점 수수료 지급 시스템을 구현하였다. 여러 보험사가 하나의 보험대리점에 수수료 파일을 지급하기에, 하나의 보험대리점 당 하나의 채널을 만드는 식으로 하였다. 또한 그 채널에는 여러 보험사가 참여할 수 있으며, 채널에 설치된 체인코드를 통해 수수료 정리에 필요한 정보들을 보내게 된다.

시스템의 구현을 위해 조직은 보험사 조직 3 개(Ins1, Ins2, Ins3)와 보험대리점 조직 2 개(Ga1, Ga2)로 구성하였다. 본 논문의 시나리오는 Ga1 이 Ins1 과 Ins2 의 보험을 판매하며 Ga2 가 Ins2 와 Ins3 의 보험을 판매한다는 가정이다. 각 보험대리점은 하나의 채널에 들어가기에, hnchannel1 에는 Ins1, Ins2, Ga1 조직이 들어가며 hnchannel2 는 Ins2, Ins3, Ga2 조직이 들어가있다. Ins2 조직의 경우 두 채널에 모두 속해있다. 구성의 형태는 그림 2 와 같다.



(Figure 2) 시스템 구성

Ga1 과 Ga2 는 데이터를 작성하면 안되기에, Ga1 과 Ga2 의 보증 정책은 Write 를 하지 못하도록 변경하였다.

본 논문에서는 하이퍼레저 패브릭 2.2.2 LTS 버전을 이용하였다. 체인코드는 TypeScript 를 사용하였으며, 데이터베이스는 CouchDB 를 이용하였다. 시스템은 HnNetwork 라는 네트워크 위에서 돌아간다. 위에서 언급하였듯 hnchannel1, hnchannel2 라는 2 개의 채널을 가지고 있으며 조직은 5 개 및 각 조직에 1 개씩의 피어가 있다. 5 개 조직 외에도 블록의 생성을 담당하는 오더러 조직이 있으며, 오더러 조직에는 2 개의 피어가 있다.

네트워크는 전부 쉘 스크립트 파일을 통해 실행된다. network.sh 파일을 통해 네트워크가 구동되며, createChannel.sh 파일을 통해 채널이 생성되고 deployCC.sh 파일을 통해 체인코드가 설치된다.

Organization	Peer port	CA port	DB port
ins1	7051	7054	5984
ins2	8051	8054	6984
ins3	9051	9054	7984
ga1	10051	10054	8984
ga2	11051	11054	9984
orderer	12051	12054	-

(Table 1) 도커 설정

각 조직들에 대한 정보는 표 1 과 같다. 이 조직들은 모두 도커를 통하여 구동되기 때문에, 도커 설정 파일에서 이와 같은 관계를 정의하였다. 또한, 오더러는 DB 를 따로 가지지 않기에 오더러에 대한 DB 포트는 존재하지 않는다.

Field	Type	Description
key	string	key value
date	string	date when executed
num	string	same as key value
companyCode	string	unique code of a company
product	string	name of a product
fee	number	fee value

(Table 2) FeeData 클래스

체인코드에 사용한 모델은 FeeData 클래스이며, 이 클래스의 형식은 표 2 와 같다. 체인코드가 실행된 날짜가 date 에 저장되며, companyCode 에는 보험사 자체 코드, product 에는 판매한 상품명, fee 에는 수수료 액수가 들어간다. 키값은 'G-{timestamp}-{random value}' 형태로 작성된다.

체인코드는 크게 4 가지 함수로 구성된다. 모든 수수료 지급 내역을 불러오는 getAll, 특정 수수료 지급 내역을 불러오는 get, 보험대리점에 전달하기 위해 지급 내역을 생성하는 create, 마지막으로 잘못된 값을 전달하였을 경우 수정을 위한 edit 로 구성된다.

```
async getAll(context: Context) {
    const startKey = '';
    const endKey = '';
    const results: FeeData[] = [];

    for await (const {key, value} of context.stub.getStateByRange(startKey, endKey)) {
        const strValue = Buffer.from(value).toString('utf8');
        let record: FeeData;
        try {
            record = JSON.parse(strValue);
            record.key = key;
        } catch(err) {
            console.log(err);
            record = new FeeData();
            record.key = key;
        }
        results.push(record);
    }
    return results;
}
```

(Figure 3) getAll

getAll 함수는 그림 3 과 같다. 파라미터는 따로 존재하지 않으며, DB 내의 모든 데이터를 읽어온다.

```
async get(context: Context, key: string) {
    const byteItem = await context.stub.getState(key);
    if ( !byteItem || byteItem.length === 0 ) {
        throw new Error(`error on loading data '${key}'`);
    }
    try {
        const fee = <FeeData> JSON.parse( byteItem.toString() );
        return fee;

    } catch(err) {
        console.log(err);
        return undefined;
    }
}
```

(Figure 4) get

get 함수는 그림 4 와 같다. 특정 키를 파라미터로 입력받아 이에 해당하는 항목이 있으면 반환해준다.

```
async create(context: Context, companyCode: string, product: string, fee: number) {
    try {
        const f: FeeData = {
            date: dateFormat(new Date(context.stub.getTxTimestamp().nanos)),
            num: this.key(context),
            companyCode: companyCode,
            product: product,
            fee: fee
        };
        f.key = f.num;
        await context.stub.putState(f.key, stateValue(f));
        console.log('new fee data generated');

    } catch(err) {
        console.log(err);
        return undefined;
    }
}
```

(Figure 5) create

create 함수는 그림 5 와 같다. 회사코드, 상품명, 수수료를 받아 새로운 데이터를 생성한다. date 는 이 함수가 실행된 순간의 밀리세컨드 값을 이용하여 Date 객체를 만든 뒤 저장한다. 상기하였듯 보험대리점에

서는 이 함수를 이용하면 안되기에, 보증 정책을 변경하여 보험사 조직들만이 이 함수를 이용할 수 있도록 하였다.

```
async edit(context: Context, key: string, field: string, value: string) {
    const byteItem = await context.stub.getState(key);
    if ( !byteItem || byteItem.length === 0 ) {
        throw new Error(`error on loading data '${key}'`);
    }
    try {
        const f = <FeeData> JSON.parse( byteItem.toString() );
        const keys = f.key.split('-');
        let sequence = 1
        if ( keys[keys.length - 1].includes('E') ) {
            sequence = parseInt( keys[keys.length - 1].substring(1) ) + 1;
            keys.length -= 1;
            keys.push('E' + sequence);
        }
        const newKey = keys.join('-');
        f.key = newKey;
        f.num = newKey;
        f[field] = value;
        await context.stub.putState(f.key, stateValue(f));
        console.log(`fee data edited`);

    } catch(err) {
        console.log(err);
        return undefined;
    }
}
```

(Figure 6) edit

edit 함수에서는 지급 내역의 수정을 할 수 있다. 파라미터로 수정할 필드와 새로 입력할 값을 받는다. 하지만 이 값은 함부로 수정되면 안되기 때문에, 수정할 경우 기존의 값을 덮어쓰는 것이 아닌 새로운 시퀀스를 달아서 쓰게 된다. 가령 기존 값이 ‘G-132-7’이었다면 이 값은 보존한 채 ‘G-132-7-1’ 값이 새로 쓰이는 것이다. 만약 이것이 한 번 더 수정된다면, ‘G-132-7-2’ 값이 새로 들어가게 된다. 이러한 방식으로 새로 시퀀스를 달아서 값을 넣게끔 한다.

위와 같은 방식을 통해 지급 내역을 각 보험사들이 통일된 규격을 통해 입력하여 관리할 수 있다. 이러한 방식을 사용할 경우 GA 에서 별도로 취합하는 방식을 사용하지 않아도 되며, 메일을 이용하여 데이터를 전송하는 등의 보안이 취약한 일 또한 발생하지 않게 된다.

4. 결론 및 향후 연구

본 논문에서는 블록체인을 이용하여 보험사에서 보험대리점에 수수료 지급 내역을 전송하는 방식에 대한 새로운 시스템을 구현하였다. 이 시스템을 통해 훨씬 더 나은 보안을 가진 관리 방식 및 체계화된 데이터 전송 방식을 기대할 수 있다.

향후 연구 계획으로는, 먼저 본 논문에서 정의한 체인코드의 FeeData 클래스 필드가 현업과 맞지 않는 부분이 다소 있을 수 있는데, 이러한 부분을 더 확실하게 다듬고자 하며, 이렇게 작성된 시스템 백엔드에 프론트엔드를 적용하여 바로 상용화 가능한 애플리케이션으로도 만들고자 한다. 이것이 실제 현업으로 확대될 경우 설치 비용은 매우 적으면서도 큰 인력 절감 효과를 기대한다.

5. Acknowledgement

이 논문은 2021 년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임(No.2018-0-00264, IoT 융합형 블록체인 플랫폼 보안 원천 기술 연구).

참고문헌

[1] “공룡 된 'GA'…보험시장 절반 삼켰다”, 매일경제 [Internet] available: https://bit.ly/3d9HLVF

[2] Nakamoto, Satoshi. Bitcoin: A peer-to-peer electronic cash system.

[3] Lamport, Leslie, Robert Shostak, and Marshall Pease. "The Byzantine generals problem."

[4] Beckert, Bernhard, et al. "Formal specification and verification of Hyperledger fabric chaincode." 3rd Symposium on Distributed Ledger Technology (SDLT-2018) co-located with ICFEM. 2018.