J. Internet Comput. Serv.
ISSN 1598-0170 (Print) / ISSN 2287-1136 (Online)
http://www.jics.or.kr

하이브리드 블록체인 기반의 안전한 펌웨어 배포 시스템

Hybrid blockchain-based secure firmware distribution system

손 민 성[1] 김 희 열[1*]
Min-sung Son Heeyoul Kim

요 약

4차 산업시대에 들어가며 IoT기기들의 수는 폭발적으로 증가하고 있다. 그에 따라 증가하는 IoT기기들의 보안사고와 비례하여 보안의 중요성에 대한 관심 또한 높아지고 있다. 하지만 IoT기기들의 제한적인 성능으로 인해 기존의 보안 솔루션을 적용하기에는 제약이 있다. 따라서 이를 해결하기 위한 새로운 자동 펌웨어 배포 솔루션을 필요로 한다. 우리는 이 문제를 해결하기 위해 퍼블릭 블록체인과 프라이빗 블록체인을 결합한 하이브리드 블록체인을 사용하는 새로운 자동 펌웨어 업데이트 시스템을 제안한다. 퍼블릭 블록체인은 다양한 펌웨어 제공업자들이 공통된 시스템을 사용하여 펌웨어 배포를 가능하게 해준다. 프라이빗 블록체인은 퍼블릭 블록체인의 트랜잭션 과부하 문제를 해결하며 IoT 기기들의 관리를 용이하게 해준다. 또한 분산 파일 저장소를 사용하여 단실실패점 없이 높은 가용성을 보장한다. 따라서 본 시스템을 사용하면 IoT 기기들의 보안 향상에 매우 효과적일 것으로 예상된다.

☞ 주제어 : 사물인터넷, 펌웨어, 블록체인

ABSTRACT

As the 4th industrial age enters, the number of IoT devices is exploding. Accordingly, the importance of security is also increasing in proportion to the increasing number of security incidents of IoT devices. However, due to the limited performance of IoT devices, there are limitations to applying existing security solutions. Therefore, a new automatic firmware distribution solution is needed to solve this problem. To solve this problem, we propose a new automatic firmware update system that uses a hybrid blockchain that combines a public blockchain and a private blockchain. The public blockchain allows various firmware providers to distribute firmware using a common system. Private blockchain solves the transaction overload problem of the public blockchain and facilitates the management of IoT devices. It also uses distributed file storage to ensure high availability without failing. Therefore, this system is expected to be very effective for improving the security of IoT devices.

☞ keyword : IoT, firmware, blockchain

1. 서 론

4차 산업시대에 들어서며 사물 인터넷(Internet of Things)을 활용한 서비스가 증가함에 따라 IoT 기기들의 수가 점점 증가하고 있다[1]. Gartner Inc. 에 따르면 2017년 IoT 기기들의 수가 2016년에 비해 31%의 증가한 약 84억개인 것으로 파악되었으며 2020에는 약 240억개로 증가할 것으로 예측하고 있다[2]. 그러나 IoT 기기들의 폭발적인 증가와는 다르게 그에 따른 보안 정책들의 표준안은 완성되지 않은 상태이며 IoT 기기들은 기본적으로 경량화 되고 낮은 성능을 가지고 있기 때문에 기존의 보안 솔루션을 적용시키는데 많은 제약을 가지고 있다.

이러한 환경에서의 IoT 기기를 보호하는 기본적인 방법중 하나는 최신 펌웨어를 설치하여 zeroday 또는 oneday 공격에 대한 저항성을 향상시키는 것 이다. 이를 위해서 펌웨어 제공업자들은 펌웨어의 무결성을 보장하며 가용성 있게 펌웨어를 제공해 줄 수 있어야 한다. 기존의 시스템은 IoT 기기들이 펌웨어 제공업자의 저장소를 통해 펌웨어를 다운받고 비대칭키 기반의 검증을 함으로써 이를 가능하게 하였다. 하지만 이러한 방식은 중간자 공격, 악의적으로 이전 버전의 펌웨어를 설치하게 하는 다운그레이드 공격, 펌웨어 제공업자를 향한 DDoS와 같은 공격에 취약하다. 또한 IoT 기기의 폭발적인 증가는 펌웨어 업데이트로 인한 네트워크의 과도한 트래픽

1 Department of Computer Science, kyonggi University, Suwon, 16227, Korea.
* Corresponding author (heeyoul.kim@kgu.ac.kr)
[Received 02 September 2019, Reviewed 23 September 2019, Accepted 15 October 2019]
☆ 이 논문은 2018년도 정부(과학기술정보통신)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No. 2018R1C1B6002903)
☆ 본 연구는 2019년 경기대학교 대학원 연구원장학생 장학금 지원에 의하여 수행되었음.

을 유발하여 네트워크 병목현상을 발생시킬 수 있으며 펌웨어 제공업자의 저장소가 IoT 기기들의 수많은 펌웨어 업데이트 요청에 응답하지 못할 수도 있다. 저장소의 가용성의 확보를 위해 서버를 증설한다 하더라도 그에 따른 비용또한 매우 높다. 따라서 이러한 문제를 해결하고 공격으로부터 안전한 펌웨어 배포 모델에 대한 논의가 필요하다.

위 문제를 해결하기 위하여 우리는 블록체인 기반의 안전한 펌웨어 배포 모델을 제안한다. 제안하는 모델은 퍼블릭 블록체인과 프라이빗 블록체인을 함께 사용하는 하이브리드 블록체인을 사용한다. 퍼블릭 블록체인은 펌웨어 제공업자가 펌웨어의 메타데이터를 배포하기 위해 사용되며 기존의 퍼블릭 네트워크를 유지하는 노드와 다양한 펌웨어 제공업체가 이 네트워크에 참가한다. 프라이빗 블록체인은 퍼블릭 블록체인으로부터 펌웨어의 메타데이터를 받아 IoT 디바이스들에게 배포하기 위해 사용되며 IoT 기기의 관리를 원하는 특적 지역(예: 스마트 팩토리, 스마트 시티)에서 구축하고 관리한다. 퍼블릭 블록체인은 펌웨어 제공업자가 업로드하는 펌웨어의 메타데이터 정보에 대한 무결성을 검증해 주며 프라이빗 블록체인은 퍼블릭 블록체인의 느린 TPS와 트랜잭션 과부화 문제를 해결해준다. 또한 허가형 블록체인이라는 특성을 통해 IoT 기기들의 보안성을 향상시켜 준다.

본 논문은 다음과 같이 구성되어 있다. 2장은 관련 연구에 대한 내용을 언급할 것 이며 3장은 제안하는 모델에 대한 자세한 설명을 언급할 것 이다. 4장은 제안하는 모델의 성능 및 목표평가를 진행하고 5장은 실제 구현을 통한 실험을 진행할 것이며 마지막으로 6장 결론으로 논문을 마친다.

2. 관련연구

2.1 기존의 IoT 기기에 대한 최신 펌웨어 업데이트 방법.

기존의 IoT 기기의 최신 펌웨어 업데이트 방법은 서버-클라이언트 구조의 업데이트 방식을 사용한다. 펌웨어 제공업자는 펌웨어를 저장하고 배포하는 저장소를 생성, 관리하며 IoT기기들은 적절한 저장소를 통해 펌웨어를 다운로드 받을 수 있다[3].

이때, 펌웨어의 무결성을 보장하기 위해서 일반적으로 ECC 또는 RSA와 같은 비대칭 암호화기법을 사용한다. 펌웨어 제공자는 펌웨어파일을 해쉬화 시킨 뒤 개인키로 서명하여 공개키, 펌웨어 파일과 함께 저장소에 배포한다. IoT기기는 저장소를 통해 펌웨어파일, 서명, 공개키를 받고 로컬에서 공개키를 이용하여 서명을 검증한 뒤 펌웨어 업데이트를 진행하게 된다.

하지만 이러한 펌웨어 업데이트방식은 수천만개의 IoT 기기들의 모든 트래픽을 감당할 수 없을 뿐만 아니라 중간자공격, 재전송공격, DDoS등에 취약하다. 따라서 폭발적으로 증가하는 IoT 기기들을 관리하기에 기존의 업데이트 방식은 가용성과 확장성을 보장하기에 어려움이 있다.

2.2 블록체인과 스마트 컨트렉트

블록체인은 2009년 Nakamoto Satoshi에 의해 처음으로 소개되었다[4]. 블록체인은 DLT(Distribute Ledger Technology)중 하나이며 어느정도의 Safty와 liveness를 보장할 뿐만 아니라 데이터의 무결성 또한 보장하는 기술이다.

기본적인 개념은 일정기간동안 발생한 거래들을 하나의 블록으로 묶고 이 전에 만들었던 블록의 해쉬값을 현재 블록에 추가하여 합의를 진행함으로서 해쉬함수의 비가역성을 이용해 연결된 블록이 서로를 검증해 주는 방식이다. 일반적으로 퍼블릭 블록체인에서는 PoW(Proof of Work)라는 합의 방식을 사용한다. 이 합의 방식은 생성하고자 하는 블록의 해쉬값이 블록체인 네트워크에서 정한 난이도 보다 낮은값을 갖게 하는 Nonce값을 찾아내게 하는 방식이다. 즉, 이 값을 찾아내는 노드가 블록을 생성하는 권한을 갖게 하는 일종의 로또방식의 합의 프로토콜이다. Nonce 값을 찾기 위해서는 높은양의 컴퓨팅 파워가 소요되며 이미 만들어진 블록체인의 내용을 바꾸기 위해서는, 즉 자신이 블록 생성 권한을 갖기 위해서는 자신의 컴퓨팅 파워가 다른 노드들 전체의 컴퓨팅 파워보다 높아야 한다. 때문에 사실상 블록체인의 내용을 변경하는 것은 불가능 하다.

블록체인의 이러한 장점을 더 활용하기 위해 Vitalik Buterin은 2015년 블록체인 시스템에 스마트 컨트렉트개념을 추가한 이더리움[5]을 발표했다. 기존의 블록체인 시스템은 트렌젝션에 간단한 연산만 가능한 튜링 불완전한 스크립트만을 지원했지만, 이더리움에서는 Solidity라는 튜링 완전한 언어를 지원하게 함으로써 블록체인에 기록된 데이터들을 이용한 프로그래밍을 가능하게 하였으며 블록체인을 이용한 다양한 서비스를 제공할 수 있도록 하였다. 이 개념은 후의 이오스, 하이퍼레저 패브릭[6]과 같은 블록체인 플랫폼에도 적용되어 사용되고 있다.

2.3 기존의 블록체인을 이용한 펌웨어 업데이트

Lee and Lee[7]가 제안하는 구조는 IoT 장치들을 블록체인 네트워크의 일반노드로서 작용하게 함으로써 블록체인 네트워크를 유지시킨다. 펌웨어 제공업자는 블록체인 네트워크에 속한 검증노드를 반드시 가지고 있어야 하며 펌웨어 파일을 배포하고 검증하기 위해 존재한다. 즉 검증노드는 언제나 최신 펌웨어 파일을 갖고 있다. 또한 일반노드들은 서로가 서로의 버전을 검증해주는 역할로서 존재한다.

일반노드가 검증노드에게 자신의 펌웨어가 최신인지에 대한 질의를 하면 검증노드는 일반노드의 버전이 최신인지 확인하고 최신이 아니라면 펌웨어 다운로드과정을 진행하게 한다. 만일 일반노드 A가 다른 일반노드 B에게 자신의 펌웨어가 최신인지에 대한 질의를 하면 일반노드 B는 일반노드 A의 버전이 자신보다 낮은지 확인한 후 자신보다 버전이 낮다면 A에게 자신의 펌웨어를 다운받을 수 있는 링크를 보내주며 그 반대라면 일반노드 A로부터 일반노드 A의 펌웨어를 받을 수 있는 링크를 받는다. 만일 둘의 버전이 같다면 자신들의 버전에 대한 검증을 블록체인 네트워크를 통해 다른 노드들에게 요청하고 PoW를 통해 6번의 확인을 요구한다.

하지만 이 방법은 노드들을 IoT 기기들로 구성하였다. IoT 기기들은 기본적으로 낮은 성능을 갖기 때문에 악의적인 사용자가 블록체인 네트워크에 들어와 PoW에 대해 51%공격을 시도한다면 펌웨어 업데이트 매커니즘이 무너질 가능성이 있다. 또한 [8]에서는 위 매커니즘이 펌웨어 검증에 6번의 확인을 요구하기 때문에 많은 IoT 기기들이 동시에 펌웨어 업그레이드를 요청할 경우 확장성의 문제가 발생할 수 있으며 펌웨어다운로드 링크를 신뢰하는 방법을 언급하지 않았기 때문에 공격자가 악성 펌웨어를 제공할 가능성도 존재한다고 말하고 있다.

[9]에서는 펌웨어 제공업자들을 도와주는 브로커 저장소의 개념을 넣음으로써 펌웨어 제공업자에게 가해지는 네트워크 트래픽을 분산시키는 방법을 제시했다. 또한 게이트웨이와 IoT 기기들을 연결시켜 IoT기기가 게이트웨이에게 펌웨어 업데이트 메시지를 전송하였을 때 게이트웨이가 블록체인 네트워크에서 펌웨어의 메타데이터를 받고 메타데이터에 있는 URL을 통해 펌웨어를 다운받아 IoT기기에게 전송하는 방안을 제시했다. 펌웨어 제공자는 펌웨어 업데이트를 위해 새로운 스마트 컨트랙트를 생성하여 블록체인 네트워크에 펌웨어의 업데이트를 알리는 방안을 제시하였고, 브로커 저장소들 또한 새로운 펌웨어의 업데이트 확인 및 다운로드를 위해 새로운 스마트 컨트랙트를 생성하여 확인하는 방안을 제시하였다.

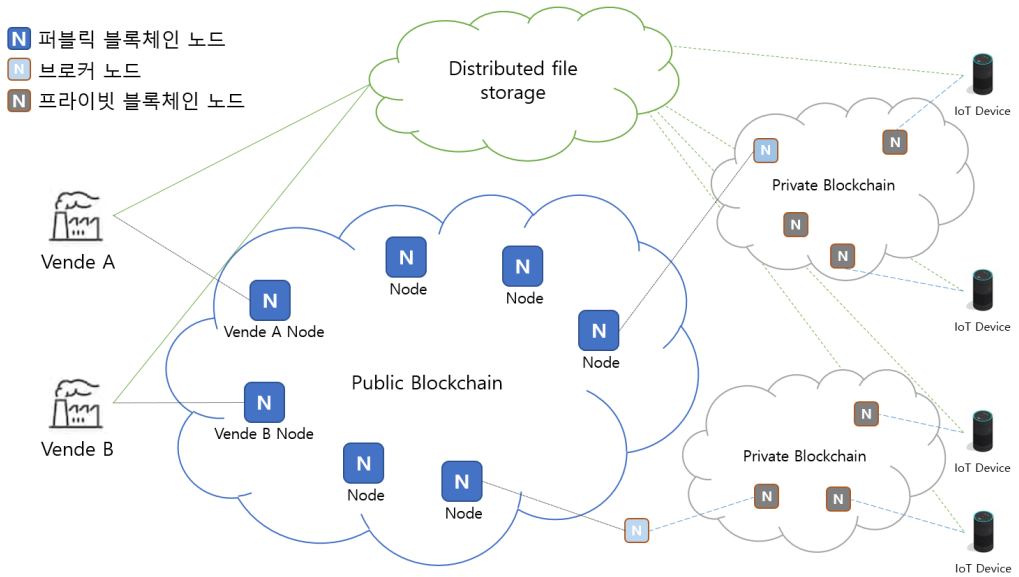
하지만 논문의 저자는 펌웨어 브로커를 누가 운영할 것인가에 대하여 서술하지 않았다. 만일 펌웨어 제공업자가 운영한다면 이는 기존의 펌웨어 업데이트 시스템과 같이 펌웨어 제공업자가 모든 트래픽과 비용을 감당하게 되는 결과를 초래한다. 또한 게이트웨이를 사용하여 IoT 기기들을 묶음으로서 트래픽 절감효과를 노린 것으로 보이나 저자는 게이트웨이에 펌웨어 캐싱등의 기술을 서술하지 않았다. 이는 결국 IoT 디바이스가 게이트웨이에 펌웨어 업데이트를 요청할 때 마다 게이트웨이가 저장소로부터 펌웨어를 다운받아야 한다는 것을 의미하며 이는 엄청난 트래픽 낭비이다. 마지막으로 펌웨어 제공자가 펌웨어를 업데이트 할 때 마다 새로운 컨트랙트를 생성하는 것은 스마트 컨트랙트의 변수를 업데이트하는 기존의 활용 방안에 비해 높은 비용이 요구되므로 비용적으로 매우 손해를 보는 일이다.

[8]에서는 쌍곡선암호를 사용한 identity기반의 암호화를 사용하여 기존의 블록체인 플랫폼보다 경량화된 서명 및 검증 방식을 사용한 허가형 블록체인을 플랫폼을 제안하였다. 허가형 블록체인이란 허가된 사용자들만이 구축하고 사용할 수 있는 프라이빗 블록체인의 일종이다. 또한 업로드되는 펌웨어을 여러 보안 위협으로부터 방어하기 위해 구글의 VirusTotal API를 사용하여 펌웨어 업로드시의 바이러스 검사를 통해 펌웨어의 보안성을 더욱 향상시켰다.

그러나 본 논문의 모델은 허가형 블록체인으로만 이루어져 있다. 따라서 추후 더욱 많은 IoT 제조업자들이 생겨나고 네트워크가 거대해 진다면, 새로운 IoT 제조업자들이 네트워크에 접근하기 위한 인증서를 발급받는데에 불편을 겪을 것이며 이는 확장성의 문제로 이어질 것이다. 또한 제안하는 모델은 PoW 합의방식을 사용하고 있다. 만일 네트워크에 점점 더 많은 제조업자들이 들어오게 되고, 그중 악의적인 사용자가 존재하며 강한 해쉬 파워를 가지고 있다고 가정하면 이 블록체인 시스템은 퍼블릭 블록체인에 비해 손쉽게 무너질 것이다.

3. 제안 모델

이번 절에서는 우리가 제안하는 안전한 펌웨어 업데이트를 위한 블록체인 기반의 시스템 모델에 대하여 논



(그림 1) 시스템 모델
(Figure 1) System model

할 것 이다.

제안하는 모델은 퍼블릭 블록체인과 프라이빗 블록체인으로 구성되어 있다. 퍼블릭 블록체인은 펌웨어 제공업자가 펌웨어를 등록하기 위해 사용한다. 퍼블릭 블록체인은 누구나 네트워크에 참여할 수 있도록 설계되었다. 이러한 퍼블릭 블록체인의 특성을 이용하여 제안하는 모델에 대한 펌웨어 제공업자들의 접근성을 향상시켰으며 다양한 펌웨어 제공업자들이 공통된 시스템으로 펌웨어 배포를 가능하게 하였다. 프라이빗 블록체인은 제안하는 구조를 사용하고자 하는 특정 조직이 구축한다. 또한 조직에 속한 IoT 기기들의 트랜잭션 이력 등을 이용하여 IoT기기들을 관리하기 위해 사용된다.

퍼블릭 블록체인의 TPS(Transaction Per Second)는 비트코인 약 7TPS, 이더리움 약 17TPS의 성능을 갖는다. 블록체인이 아무리 가용성이 보장한다 할지라도 수천만대의 IoT기기들이 펌웨어 업데이트 트랜잭션을 동시에 전송하게 되면 트랜잭션의 처리시간이 비약적으로 증가할 것 이다. 따라서 우리모델은 프라이빗 블록체인이 퍼블릭 블록체인에 대하여 일종의 캐싱을 함으로써 IoT 기기들이 자신이 속한 프라이빗 블록체인을 통해 업데이트를 진행할 수 있도록 하였다. 프라이빗 브록체인의 TPS는 수천TPS를 넘나들기 때문에[10] 프라이빗 블록체인에 속한 IoT 기기들이 동시에 많은 업데이트 트랜잭션을 발생시키더라도 즉각적으로 반응할 수 있다. 또한 허가형 블록체인이기 때문에 외부의 공격으로부터 면역이 있고 IoT기기들의 트랜잭션 로그들이 모두 남기 때문에 관리적 측면에서도 이득을 취할 수 있다.

본 모델은 아래의 보안 및 성능 목표를 갖는다.

- 펌웨어 무결성 : 제안하는 모델은 펌웨어의 변조에 대한 내성이 있어야 하며 항상 올바른 데이터를 제공해 줄 수 있어야 한다.
- DDos 저항성 : 제안하는 모델은 DDos공격으로 인한 펌웨어 업데이트의 지연이 없어야 하며 단일 실패점으로 인한 업데이트 실패가 없어야 한다.
- 확장성 : 제안하는 모델은 확장성이 좋아야 한다.
- 비용 : 제안하는 모델은 비용적 측면에서의 효율성이 좋아야 한다.

3.1 시스템 모델

제안하는 모델은 다음과 같은 노드들로 이루어져 있다. 노드는 어느 조직 및 개인이 운영하는 서버일 수도 있고 aws나 azure와 같은 클라우딩 컴퓨터를 사용할 수도 있다.

1. 퍼블릭 블록체인 노드 : 퍼블릭 블록체인을 구성하는 노드이며 기존의 퍼블릭 블록체인을 이루는 노드와 여러 펌웨어 제공업자들의 노드들로 이루어져 있다. 펌웨어 제공업자는 자신의 노드를 퍼블릭 블록체인의 노드로 참가하여 사용해야 한다.
2. 프라이빗 블록체인 노드 : 본 모델를 사용하고자 하는 특정 지역이 구성하는 노드이며 프라이빗 블록체인을 구성한다.
3. 브로커 노드 : 퍼블릭 블록체인과 프라이빗 블록체인을 연결시켜주는 노드를 의미한다. 브로커 노드는 퍼블릭, 프라이빗 블록체인의 노드 역할을 수행할 수 있으며 원한다면 동시에 수행할 수도 있다. 단일 실패점을 피하기 위해 여러대의 브로커노드를 설치하는 것을 권장하며 프라이빗 블록체인의 펌웨어 업로드 및 펌웨어 제공업자 등록 권한을 갖는다. 브로커 노드는 프라이빗 네트워크를 구축하는 지역에서 유지한다.

브로커 노드는 프라이빗 블록체인의 렛저 업데이트 권한을 갖기 때문에 모든 노드중에서 가장 중요하다. 따라서 HSM(hardware secrity module)을 사용하는 등 보안 모듈 및 기법을 적용하여 프라이빗 네트워크에 접속할 수 있는 개인키를 안전하게 보관해야 한다.

4. IoT 기기 : IoT 기기는 자신이 속한 특정 지역의 프라이빗 네트워크에 연결되며 블록체인 네트워크에 노드로써 직접적으로 참여하지 않고 클라이언트로서 연결된다.

3.2 설정 및 등록

제안하는 모델을 사용하기 위해서는 일련의 설정 및 등록 과정이 필요하다. 본 모델에서는 퍼블릭 블록체인과 프라이빗 블록체인이 동시에 사용되기 때문에 이를 펌웨어 제공업자(퍼블릭)와 펌웨어 요청자(프라이빗)의 관점으로 나누어 논해볼 것 이다. 사용되는 표기법은 다음과 같다.

(표 1) 파라메터 표기법
(Table 1) Natation Description

Notation	Description
V	Vendor name
m	Model Type
v	Firmware Version
u	URI
chk	Firmware Checksum
$h()$	Hash fucntion
D	Digest. D = $h(V,m,v,u,chk)$
σ	Signature
Pub_v	Vendor V's Public key
$FMstruct$	Struct{ V,m,v,u,chk,D_{σ_v}}

퍼블릭 블록체인과 프라이빗 블록체인을 동시에 사용하므로 각 네트워크에서 동작하는 스마트 컨트랙트도 다르게 존재한다. 퍼블릭 블록체인의 스마트 컨트랙트를 Pub_SC_Firmware, 프라이빗 블록체인의 스마트 컨트랙트를 Prv_SC_Firmware라고 정의하며 그 구조는 아래와 같다.

(표 2) Pub_SC_Firmware의 인터페이스
(Table 2) Interface of Pub_SC_Firmware

Contract name : Pub_SC_Firmware
Interface : setVendor(V, Pub_V) returns (bool) updateFirmware(V,m,v,u,chk,D_{δ_V}) returns (bool) getFirmware(V,m) returns (V,m,v,u,chk,D_{δ_V})

(표 3) Prv_SC_Firmware 인터페이스
(Table 3) Interface of Prv_SC_Firmware

Contract name : Prv_SC_Firmware
Interface: : setVendor(V, Pub_V) returns (bool) updateFirmware(V,m,v,u,chk,D_{δ_V}) returns (bool) cehckFirmware(V,m,c) returns (V,m,v,u,chk,D_{δ_V})

또한 컨트랙트 A의 함수 B를 A.B로 정의할 것 이다. 예를 들면 Prv_SC_Firmware 컨트랙트의 setVendor함수는 Priv_SC_Firmware.setVendor() 로 표기된다.

3.2.1 펌웨어 제공업자

펌웨어 제공업자가 퍼블릭 블록체인에 펌웨어를 등록하기 위해서는 먼저 Pub_SC_Firmware.setVendor() 를 실행시킴으로써 자기 자신을 등록하여야 한다. 일반적으로 퍼블릭 블록체인의 Account을 생성함으로써 개인키와 공개키 쌍을 제공받으며 이를 이용하여 퍼블릭 블록체인에

자신임을 증명할 수 있고, 데이터에 서명을 함으로써 무결성을 보장할 수 있다.

(표 4) setVendor 알고리즘
(Table 4) Algorithm of setVendor

Contract : Pub_SC_Firmware, Prv_SC_Firmware
function name : setVendor
Input : V, Pub_V Output : bool Variable : vendor_map //mapping V to Pub_V Algorithm : IF V exist in vendor_map : return false ELSE : vendor_map[V] = Pub_V return true

펌웨어 제공업자가 Pub_SC_Firmware에 자신을 등록하기 위해 Pub_SC_Firmware.setVendor()를 실행시키면 컨트랙트는 펌웨어 제공업자의 이름이 이미 등록되어있는지 확인 후 등록되어 있다면 flase, 등록되어 있지 않다면 vendor_map에 펌웨어 제공업자의 공개키를 저장한 뒤 true를 반환한다. 이 값은 블록체인에 저장되므로 블록체인의 특징인 무결성으로 인해 Pub_V가 V의 Pulbic Key임을 보장받는다.

브로커 노드는 제공업자의 등록 이벤트를 감지한다. Pub_SC_Firmwrae.setVendor()의 실행이 감지되면 브로커 노드는 새로 등록된 펌웨어 제공업자의 정보를 이용하여 Prv_SC_Firmware.setVendor()를 실행시킨다. 이 과정를 통해 프라이빗 블록체인에도 펌웨어 제공업자들의 공개키가 업데이트된다.

3.2.2 펌웨어 요청자

펌웨어 요청자는 본 모델을 사용하여 IoT 기기들을 관리하길 원하는 특정 지역을 의미한다. 프라이빗 블록체인은 펌웨어 요청자가 구축하는 프라이빗 블록체인 네트워크이며 보안을 위해 CA(certification Authority)를 통한 인증서 기반의 허가형 블록체인을 사용해야 한다. 따라서 IoT 기기들은 구축된 프라이빗 블록체인 플랫폼을 사용하기 위해서 CA로부터 인증서를 발급받아 자신이 허가받은 기기임을 증명해야 한다.

펌웨어 요청자는 브로커 노드를 유지하여야 한다. 브로커 노드는 퍼블릭 블록체인과 프라이빗 블록체인을 이어주는 역할을 하며 프라이빗 블록체인의 펌웨어 제공업자 등록 및 최신 펌웨어 정보 업데이트 권한을 가지고 있다. 브로커 노드가 죽으면 프라이빗 네트워크는 더 이상 퍼블릭 네트워크와 통신할 수 없기 때문에 브로커 노드를 여러개 두어 단일 실패점을 제거 해야한다.

3.3 펌웨어 배포

제안하는 모델은 프라이빗 블록체인이 퍼블릭 블록체인으로부터 데이터를 받아와 자신의 블록을 업데이트하는 방식을 취하고 있다. 따라서 펌웨어 업로드 과정은 퍼블릭과 프라이팟 블록체인 두 단계로 진행된다.

일반적으로 퍼블릭 블록체인에서는 트랜젝션을 발생시키는데에 있어 용량제한을 두거나 수수료를 부과하여 큰 데이터의 업로드를 지양한다. 이는 트랜잭션에 너무 큰 데이터가 들어있을 경우 네트워크 오버헤드가 커질 뿐만 아니라 모든 블록을 저장해야 하는 마이너들에게 큰 부담이 되기 때문이다. 따라서 본 모델에서는 위 문제를 해결하기 위해 펌웨어 파일 전체를 블록체인 네트워크에 전파하지 않고 펌웨어 파일의 메타데이터만 블록체인에 저장한다.

펌웨어 파일을 블록체인 네트워크에 올리지 않기 때문에 우리는 IPFS[11], BitTorrent[12]와 같은 분산 파일 저장 플랫폼을 사용하여 펌웨어를 업로드 한다. 이것을 사용함으로써 우리는 펌웨어 다운로드의 단일 실패점 문제 및 DDos공격에 대한 저항 그리고 확장성의 향상을 꾀할 수 있다. 분산 파일 저장 플랫폼을 이용하여 파일을 업로드 하게 되면 파일을 다운로드 받을 수 있는 링크 u 를 얻을 수 있다. 이 링크를 펌웨어 파일의 메타데이터로 넣어주고 블록체인을 통해 이 주소의 무결성을 보장해 준다면 IoT 기기들이 이 링크를 통해 안전하게 펌웨어를 다운받을 수 있게 된다.

이제 이 절에서는 퍼블릭 블록체인에 펌웨어의 메타데이터를 업데이트하고 퍼블릭 블록체인으로부터 펌웨어의 메타데이터를 받아와 프라이빗 블록체인에 업데이트 하는 과정이 어떻게 진행되는지에 대해 서술한다 .

3.3.1 펌웨어 제공업자

펌웨어 제공업자가 새로운 펌웨어를 배포하는 과정은 다음과 같다.

먼저 업로드 할 최신 펌웨어의 메타데이터 $[V,m,v,u,chk,D_{\delta_V}]$를 이용하여 트랜젝션을 생성하고 Pub_SC_Firmware.updateFirmware()를 실행시킨다.

(표 5) updateFirmware 알고리즘
(Table 5) Algorithm of updateFirmware

Contract : Pub_SC_Firmware
function name : updateFirmware
Input : V,m,v,u,chk,D_{σ_V}, Output : bool Variable : vendor_map // mapping V to Pub_V firmware_map // mapping $h(V,m)$ to $FMstruct$ Algorithm : IF V not exist in vendor_map : return false ELSE IF vendor_map[V] != sender.Pub : return false ELSE IF firmware_map[$h(V,m)$].v >= v : return false ELSE : firmware_map[$h(V,m)$] = new $FMstruct$ return true

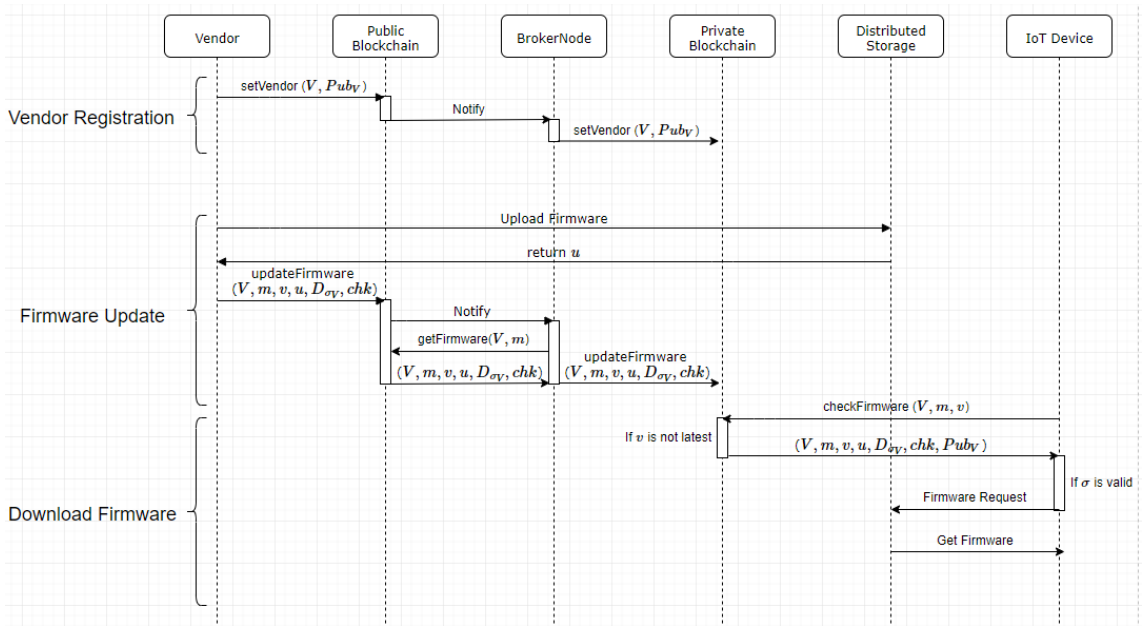
전송된 데이터는 컨트랙트를 통해 보낸 펌웨어 제공업자가 등록되어 있는지, 트랜젝션을 보낸 사용자가 V가 맞는지, 이미 등록한 펌웨어의 버전보다 높은 버전의 펌웨어가 맞는지, 서명이 올바른지를 검증하고 블록에 업데이트 함으로써 데이터의 무결성을 보증한다.

3.3.2 브로커 노드

브로커 노드는 퍼블릭 블록체인에 연결되어 Pub_SC_Firmware.updateFirmware()의 실행을 감지한다. 실행이 감지되면 업데이트된 펌웨어의 V,m를 이용하여 Pub_SC_Firmware.getFirmware()를 실행시켜 최신 펌웨어의 메타데이터를 받아온다. 그런 뒤 받아온 메타데이터를 이용하여 Prv_SC_Firmware.updateFirmware()를 실행시켜 프라이빗 블록체인에 최신 펌웨어의 메타데이터를 업데이트 한다.

(표 6) getFirmware 알고리즘
(Table 6) Algorithm of getFirmware

Contract : Pub_SC_Firmware
function name : getFirmware
Input : V,m Output : V,m,v,u,chk,D_{σ_V} Variable : firmware_map // mapping $h(V,m)$ to $FMstruct$ Algorithm : IF firmware_map[$h(V,m)$] is exist : return V,m,v,u,chk,D_{σ_V} ELSE : return nill



(그림 2) 전체 흐름도
(Figure 2) whole flow

(표 7) updateFirmware 알고리즘
(Table 7) Algorithm of updateFirmware

Contract : Prv_SC_Firmware
function name : updateFirmware
Input : V,m,v,u,chk,D_{σ_V} Output : bool Variable : firmware_map // mapping $h(V,m)$ to $FMstruct$ Algorithm : IF sender is not Broker : return false ELSE IF firmware_map[$h(V,m)$].v >= v : return false ELSE IF σ_V is not valid reutrn false ELSE : firmware_map[$h(V,m)$] = new $FMstruct$ return true

Prv_SC_Firmware.updateFirmware()의 알고리즘은 표 7과 같으며 스마트 컨트랙트에 데이터가 전송되면 트랜젝션 생성자가 브로커 인지, 이미 등록한 펌웨어의 버전보다 높은 버전의 펌웨어가 맞는지, 서명이 올바른지를 검증하고 블록에 업데이트 함으로써 데이터의 무결성을 보증한다.

3.2 펌웨어 다운로드

이 절에서는 IoT 디바이스가 프라이빗 블록체인으로부터 펌웨어를 다운로드받는 과정에 대하여 서술한다.

IoT 기기는 그 특성상 지속적인 네트워크연결을 항상 보장받지 못한다. 따라서 블록체인 네트워크에 주기적으로 트랜젝션을 전송해 자신이 최신 업데이트를 가지고 있는지 확인하는 함수인 Prv_SC_Firmware.checkFirmware()를 실행함으로써 자신의 펌웨어 버전을 확인하고 업데이트 한다.

펌웨어의 버전이 최신이 아니라 판단되면 IoT 기기는 프라이빗 블록체인 네트워크로부터 펌웨어의 메타데이터와 펌웨어 제공업자의 공개키를 받을 수 있다. 메타 데이터와 공개키를 받은 IoT 기기는 서명을 검증하여 서명이 올바르다면 메타데이터에 포함된 URI를 이용하여 IPFS, BitTorrent와 같은 분산 저장 플랫폼을 통해 안전하게 펌웨어를 다운받을 수 있다.

(표 8) checkFirmware 알고리즘
(Table 8) Algorithm of checkFirmware

Contract : Prv_SC_Firmware
function name : checkFirmware
Input : V,m,v Output : V,m,v,u,chk,D_{σ_V} Variable : firmware_map // mapping $h(V,m)$ to $FMstruct$ vendor_map // mapping V to Pub_V Algorithm : IF firmware_map[$h(V,m)$] is not exist : return nil ELSE if firmware_map[$h(V,m)$].$v \leq v$: return nil ELSE : Pub = vendor_map[V] return V,m,v,u,chk,D_{σ_V},

4. 성능 및 목표 평가

4.1 펌웨어 무결성

사토시 나카모토는 비트코인 백서에서 블록이 6개 이상 생성되면 공격자가 블록의 내용을 변경하는 것이 0.1% 이하라는 것을 수학적으로 증명하였다. 또한 현재 대부분의 퍼블릭 블록체인의 합의방식이 PoW이며 이들은 6번의 컨펌을 기다린 뒤 커밋된다. 따라서 퍼블릭 블록체인 네트워크에 전파한 펌웨어 메타데이터가 변조될 확률은 매우 낮다.

또한 우리는 펌웨어의 무결성을 보장하기 위해 펌웨어 파일의 체크섬을 메타데이터로 저장하고 이를 개인키로 서명한다. 위에서 메타데이터의 변조 확률이 매우 낮음을 알았기 때문에 우리는 이 체크섬을 신뢰할 수 있고, 서명을 검증하고 다운받은 펌웨어 파일을 체크섬과 비교함으로써 펌웨어의 무결성을 확신할 수 있다.

4.2 DDoS 저항성

우리가 제안하는 모델은 기본적으로 p2p(peer to peer) 시스템을 기본으로 하고 있다. 펌웨어 파일의 메타데이터는 블록체인을 통해 전파되고 펌웨어 파일은 분산 파일 저장소를 통해 배포되기 때문에 단일 실패점이 없다. 또한 우리 모델은 단일 실패점을 제거하기 위해 여러개의 브로커 노드를 구축하는 것을 권장하고 있다. 따라서 제안하는 구조는 DDoS에 저항성을 가지고 있으며 가용성 또한 충족한다.

(표 9) 구현 환경
(Table 9) Implementation environment

컴포넌트	사용 플랫폼	구동 환경	특이사항
퍼블릭 블록체인	이더리움	windows10 Intel i5-7400 16GB 64bit	Dev mode
프라이빗 블록체인	하이퍼레저 페브릭	ubuntu 16.04 i5-8400 4GB 64bit	docker
브로커 노드	nodejs + web3	ubuntu 16.04 i5-8400 4GB 64bit	
분산 파일 저장소	IPFS	ubuntu 16.04 i5-8400 4GB 64bit	
IoT Device	nodejs + fabric sdk	raspberry pi3 Model B+	

4.3 확장성

우리가 제안하는 모델은 펌웨어 제공업자들이 펌웨어 파일을 제공하기 위해 퍼블릭 블록체인과 분산 파일 저장소를 사용한다. 이들은 기본적으로 p2p를 기반으로 하기 때문에 사용자들이 늘어나더라도 특정 단말기에 부하가 집중되지 않는다. 또한 퍼블릭 블록체인을 사용하기 때문에 펌웨어 제공업자들이 본 모델을 사용하는데에 있어서 제약이 없다.

또한 프라이빗 블록체인은 수천만대의 IoT 기기들이 퍼블릭 블록체인에 펌웨어 업데이트 요청 트랜젝션을 전송함으로써 퍼블릭 블록체인에 과부화가 걸리는 것을 방지한다. 따라서 본 모델은 확장성 있다고 할 수 있다.

4.3 비용

(표 10) 시나리오별 비용
(Table 10) Cost per scenarios

시나리오	Gas	Eth	Won
스마트 컨트랙트 등록	1,441,364	0.02883	₩6520.7
펌웨어 제공자 등록	43920	0.000043920	₩9.9390
펌웨어 등록	199922	0.000199922	₩45.197
펌웨어 다운로드	0	0	₩0

구현된 모델을 기반으로 2019년 8월 시점, 각 시나리오별 가스 사용량을 계산하였다. 스마트 컨트랙트는 한번만 등록되면 모든 펌웨어 제공자가 사용 가능하므로 실질적으로 펌웨어 제공자 등록, 펌웨어 등록 및 다운로드 시나리오의 가스량만 확인하면 된다. 실험결과 매우 적은 양의 비용으로 시나리오가 진행되는 것을 볼 수 있다.

5. 구현 및 실험

이번 절에서는 구현된 구조를 기반으로 펌웨어 제공업자의 펌웨어 업로드부터 IoT 기기의 펌웨어 다운로드까지의 전체적인 흐름을 보인다. 구현된 환경은 표 9와 같다.

5.1 펌웨어 제조업자 등록.

```
function setVendor(...) ... {
  if(vendor[_vendorName].isValue) {
    return false;
  }

  vendor[_vendorName] = Address(msg.sender, true);
  emit newVendor(_vendorName, msg.sender);
  return true;
}
```

(그림 3) setVendor solidity 소스코드
(Figure 3) Solidity source code of setVendor

컨트랙트 소스를 보면 newVendor 이벤트를 사용하여 새로운 펌웨어 제공업자 등록을 알린다. 브로커 노드들은 위 이벤트를 수신하고 자신이 속한 프라이빗 블록체인에 새로운 펌웨어 제공업자에 대한 정보를 업데이트 한다.

브로커 노드는 이더리움 노드와 웹소켓으로 연결되어 newVendor 이벤트를 수신한다. 이때 web3의 subscribe 기능을 적절히 사용하면 특정 컨트랙트에서 발생하는 특정 이벤트를 실시간으로 수신받고 그에 따른 핸들러를 작성할 수 있다. 본 구현에서는 프라이빗 블록체인으로의 트랜젝션 전송을 핸들러로 작성하였다.

```
{
  _id: 0x4e8670d1bc1e1ba603f324678383f120343e7616356ec4d07ff6e8a5522b6739,
  _rev: 1-65c40fea28a8799e6845e17eeccb3d44,
  digest: 0x29167832d11269392b407c091eba6aeeaee9dbd3d82364a395c9ac0f10d72aac,
  vendor_name: Vendor,
  pubkey: 0x7cafe827c07faf145ec64265f75421e17e40395f,
  version: 1,
  ~version: u0000CgMBDQA=
}
```


(그림 4) 하이퍼레저 패브릭의 펌웨어 제공업자 정보
(Figure 4) Vendor Information for Hyperledger Fabric

couchdb를 사용하여 하이퍼레저 페브릭의 렛저를 확인한 결과 setVendor 컨트랙트의 실행으로 프라이빗 네트워크인 하이퍼레저 패브릭의 렛저에도 펌웨어 제공업자 Vendor에 대한 정보가 들어간 것을 확인할 수 있다.

5.2 펌웨어 배포

펌웨어 배포 어플리케이션은 nodejs를 기반으로 만들어졌다. 사용자가 새로운 펌웨어를 업로드 하면 어플리케이션은 새로운 펌웨어의 버전이 기존의 버전보다 높은지 판단하고 펌웨어는 IPFS에 업로드 한다. 그런 뒤 IPFS로부터 생성되는 CID(Content Identifiers)를 펌웨어의 메타데이터 u로서 사용한다. 서명시 사용되는 개인키는 web3의 decrypt기능을 사용하여 사용자의 이더리움 keystore로부터 개인키를 추출하여 사용한다.

```
function updateFirmware(...) ... {
  if(msg.sender != vendor[_vendor].addr){
    return false;
  }

  Metadata memory fmd = Metadata(_vendor, _model,
_version, _uri, _chk, _digest, _signature);

firmware[keccak256(abi.encodePacked(_vendor, _model))]
= fmd;
  emit newFirmware(_vendor, _model);
  return true;
}
```

(그림 5) updateFirmware solidity 코드
(Figure 5) Solidity source code of updateFirmware

updateFirmware 컨트랙스 소스에도 setVendor 함수와 비슷하게 newFirmware라는 이벤트가 존재한다. 브로커 노드들은 위 이벤트를 수신하고 자신이 속한 프라이빗 블록체인에 새로운 펌웨어 정보를 등록할 것 이다.


```
{
  _id: 0x150b246c661d36fb4062087d6e8ada8f9d56530fa09c95668c1e9cb02092de57,
  _rev: 1-19788cfe04182e05960901fc3447bde6,
  digest: 0x8b1e65d965924a4bdd82ec28c737072d281cdf5c945226df3ff9d332fdfbef44,
  model: test,
  signature: 0xf135daa8767f7157efe0987d2d16443d554b42ed96dd76af208597e03b6ff56
93d10791eab0535a425abe16431b,
  uri: QmVmhVnWLGpZ7Ke7NKErxvKLSmWGfxNMR7zaAzeuw2fBJz,
  vender: Vendor,
  version: 1.0,
  ~version: u0000CgMBDAA=
}
```


(그림 6) 하이퍼레저 패브릭의 펌웨어 메타데이터
(Figure 6) Firmware metadata for Hyperledger Fabric

펌웨어 제조업자 등록절차와 마찬가지로 update Firmware의 실행으로 인해 하이퍼레저 패브릭의 렛저에도 펌웨어에 대한 정보가 업데이트 된 것을 볼 수 있다.

5.3 펌웨어 다운로드

펌웨어 다운로드는 프라이빗 블록체인에 연결된 IoT 기기에서로부터 발생한다. 프라이빗 블록체인은 하이퍼레저 패브릭을 기반으로 구축되었으며 스마트 컨트랙트는 nodejs를 기반으로 작성되었다. 컨트랙트상에서 모든 사항에 문제가 없다면 IPFS를 통해 펌웨어 파일을 다운로드 받고 서명에 문제가 없다면 업데이트를 진행하게 된다.

```
Saving file(s) to fimrware
 2.08 MiB / 2.08 MiB [=========================
islab@islab-POECB0A1-Samsung-DeskTop:/firm$ ls
fimrware
```

(그림 7) 다운로드 된 펌웨어 파일
(Figure 7) Downloaded firmware file

```
async checkUpdate(stub, args) {
  let vendor = args[0];
  let model = args[1];
  let version = args[2];

  const VMHash = sha3(String(vendor)+String(model));
  const metadata = await stub.getState(VMHash);
  if(!metadata || metadata.length === 0) {
        throw new Error(...);
  }

  const obj = JSON.parse(metadata.toString())
  if(obj.version <= version) {
        throw new Error(...);
  }

  return metadata;
}
```

(그림 8) checkUpdate 스마트 컨트랙트 소스코드
(Figure 8) Smart Contract source code of checkUpdate

6. 결 론

우리는 퍼블릭 블록체인과 프라이빗 블록체인을 연결한 안전한 펌웨어 배포 시스템을 구축하였다. 블록체인에는 펌웨어의 메타데이터를 전파하여 그 무결성을 검증할 수 있게 하였으며 스마트 컨트랙트를 이용하여 버전검증 및 배포자 검증도 가능하게 하였다. 또한 펌웨어 파일을 업로드 하기 위해 IPFS, BitTorrent와 같은 분산 파일 저장소를 사용하였으며 이를통해 가용성을 향상시켰다. 또한 IoT 기기들을 프라이빗 블록체인으로 묶음으로써 관리자가 IoT 기기들의 관리를 가능하게 하였으 퍼블릭 블록체인의 트랜잭션 과부화에 대한 문제점을 해결할 수 있었다.

따라서 제안한 구조를 사용하면 기존의 펌웨어 업데이트 시스템의 문제점인 가용성과 확장성에 대한 문제를 해결할 수 있으며 중간자 공격, 버전 다운그레이드 공격 등에 대한 저항성을 가질 수 있기 때문에 안전한 펌웨어 업데이트가 가능할 것으로 사료된다.

참고문헌(Reference)

[1] Lee Hyeon-gyu, Smart Factory Industry and Market Trends, Korea Technology Transfer Centre for National R&D Programs, 2018.
http://www.itfind.or.kr/admin/getFile.htm?identifier=02-001-180228-000004

[2] Gartner. “Gartner says 8.4 billion connected “things” will be in use in 2017, up 31 percent from 2016“, Gartner Newsroom, 2017.
https://www.gartner.com/newsroom/id/3598917.

[3] Choi B-C, Lee S-H, Na J-C, Lee J-H, “Secure firmware validation and update for consumer devices in home networking”, IEEE Trans Consum Electronics, Vol.62(1), pp.39-44, 2016.
http://dx.doi.org/10.1109/TCE.2016.7448561

[4] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System”, 2008.
https://bitcoin.org/bitcoin.pdf.

[5] G. Wood, “ETHEREUM: A secure decentralised generalised transaction ledger”, 2014.
https://gavwood.com/paper.pdf

[6] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich et al., “Hyperledger fabric: A distributed operating system for permissioned blockchains.”, 2018.
http://dx.doi.org/10.1145/3190508.3190538

[7] B. Lee, J.-H. Lee, “Blockchain-based secure firmware update for embedded devices in an internet of things environment”, J. Supercomput, pp.1-16, 2016
http://doi.org/10.1007/s11227-016-1870-0

[8] Jen-Wei Hu, Lo-Yao Yeh, Shih-Wei Liao, Chu-Sing Yang, “Autonomous and malware-proof blockchain-based firmware update platform with efficient batch verification for Internet of Things devices”, Computers & Security, Vol.86, pp.238-252, 2019.
https://doi.org/10.1016/j.cose.2019.06.008

[9] Alexander Yohan, Nai-Wei Lo, “An Over-the-Blockchain Firmware Update Framework for IoT Devices”, 2018 IEEE Conference on Dependable and Secure Computing (DSC), 2018.
http://dx.doi.org/10.1109/DESEC.2018.8625164

[10] Parth Thakkar , Senthil Nathan , Balaji Viswanathan, “Performance Benchmarking and Optimizing Hyperledger Fabric Blockchain Platform”, 2018 IEEE 26th International Symposium on MASCOTS, 2018.
http://dx.doi.org10.1109/MASCOTS.2018.00034

[11] Juan Benet, “IPFS - Content Addressed, Versioned, P2P File System (DRAFT 3)”, 2017.
https://github.com/ipfs/papers

[12] J. Pouwelse, P. Garbacki, D. Epema, H. Sips, ″The bittorrent P2P file-sharing system: Measurements and analysis″, Proc. IPTPS, 2005.
https://doi.org/10.1007/11558989_19

◐ 저 자 소 개 ◑

손 민 성(Min-sung Son)
2019년 경기대학교 컴퓨터과학과(공학사)
2019년~현재 경기대학교 대학원 컴퓨터과학과(공학석사)
관심분야 : 정보보호, 블록체인
E-mail : sns2831@gmail.com

김 희 열(Heeyoul Kim)
2000년 한국과학기술원 전산학과(공학사)
2002년 한국과학기술원 전산학과(공학석사)
2007년 한국과학기술원 전산학과(공학박사)
2009년 ~ 현재 경기대학교 컴퓨터공학부 부교수
관심분야 : 정보보호, 블록체인
E-mail : heeyoul.kim@kgu.ac.kr