경량 블록체인을 위한 경량 해시함수 Spongent의 Javascript 구현

강예준*, 김현지*, 김원웅*, 권혁동*, 서화정*†

*한성대학교 (대학원생)

*†한성대학교 (교수)

Javascript implementation of lightweight hash function Spongent for lightweight blockchain

Yea-Jun Kang*, Hyun-Ji Kim*, Won-Woong Kim*, Hyeok-Dong Kwon*, Hwa-Jeong Seo*†

*Hansung University(Graduate student)

*Hansung University(Professor)

요 약

최근 블록체인을 사물인터넷에 적용시키려는 연구가 다수 이루어지고 있다. 하지만 블록체인은 공개키 발급, 합의 알고리즘, 브로드캐스트 과정 등과 같은 작업으로 인해 많은 양의 컴퓨팅 성능을 요구한다. 따라서 블록체인을 사물인터넷에 적용하기 위해서는 블록체인 경량화 기법이 요구된다. 본 논문에서는 일반적으로 블록체인에서 사용되는 해시함수 SHA-256을 경량 해시함수인 Spongent를 사용함으로써 블록체인을 경량화하는 기법을 제안한다. 또한 Javascript를 통해 Spongent 해시함수가 사용되는 프로토타입의 블록체인을 구현하였다.

I. 서론

강력한 보안 성능을 지니고 있는 블록체인은 비교적 보안 성능이 약한 사물인터넷의 (Internet of Things : IoT) 문제점을 보완해줄 수 있다. 이러한 이유로 최근 사물인터넷 상에 블록체인을 적용하려는 연구가 다수 이루어지고 있다. 하지만 사물인터넷의 경우 저장 공간 부족, 낮은 컴퓨팅 성능 등과 같은 문제점이 존재하므로, 블록체인을 사물인터넷에 적용시키기 위해서는 블록체인을 경량화 시킬 필요가 있다. 일반적으로 블록체인에서 사용되는 해시함수는 SHA-256 해시함수를 사용한다. 하지만 해당 해시함수는 경량 해시함수에 비해 많은 양의 자원을 요구한다. 본 논문에서는 사물인터넷에 블록체인을 적용시키기 위해, Javascript를 통해 블록체인에서 사용되는 해시함수를 경량 해시함수인 Spongent를 사용하여 프로토타입의 블록체인을 구현하였다.

II. 관련 연구

2.1 경량 블록체인

기존 블록체인은 공개키 발급, 머클트리 구조, 작업증명, 브로드캐스트 등과 같은 작업으로 인해 많은 양의 컴퓨팅 성능을 요구한다. 따라서 이와 같은 블록체인을 사물인터넷 상에서 활용하기에는 사물인터넷의 부족한 자원으로 인해 매우 제한적이다. 일반적으로 블록체인에서는 SHA-256 해시함수가 사용된다. 하지만 해당 해시함수는 경량 해시함수에 비해 많은 양의 자원을 요구한다는 문제점이 존재한다. 그림 1은 프로토타입으로 구현된 기존 블록체인의 구조이다. 본 논문에서는 블록체인에서 사용되는 해시함수를 경량 해시함수인 Spongent를 사용함으로써 블록체인을 경량화 시켰다.

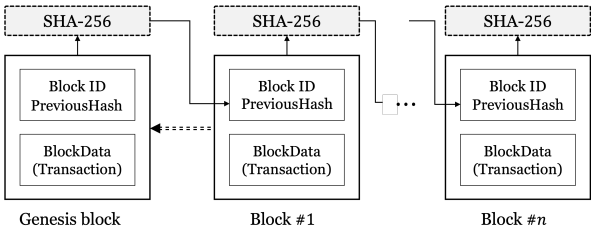

Fig. 1 Prototype original blockchain architecture

2.2 Spongent

Spongent란 스펀지 구조를 기반으로, PRESENT 타입의 permutation 연산을 수행하는 경량 해시함수이다[2]. 스펀지 구조는 크게 Absorbing 단계와 Squeezing 단계로 나뉜다. Absorbing 단계에서는 입력 데이터에서 앞쪽 r개의 비트를 모아 XOR 연산을 수행한다. 모든 입력 데이터가 처리된 후에, Squeezing 단계가 진행된다. 해당 단계에서는 데이터에서 앞쪽 r개의 비트를 결과값으로써 return한다. Spongent는 보안 수준에 따라 총 다섯 개의 종류로 나뉘며 각각 88, 128, 160, 224, 256 bits의 해시값을 출력한다.

2.3 Javascript

Javascript는 웹페이지를 동적으로 만들기 위해 만들어진 객체 기반 프로그래밍 언어이다[3]. 따라서 주로 웹 브라우저에서 사용되며, 경우에 따라 Node.js와 같은 프레임워크를 사용하여 서버측에서도 사용 가능하다. Javascript는 타입을 명시하지 않는 인터프리터 언어이기 때문에, 컴파일 타입 언어에 비해 실행 속도가 느리다는 단점이 있다. 하지만 최근 실행 중에 코드를 컴파일하는 JIT (Just In Time Compiler) 컴파일러를 통해 속도를 높여 이와 같은 문제점을 해결하였다.

III. 본론

본 논문에서는 Javascript를 통해 Spongent 해시함수가 적용된 프로토타입의 블록체인을 구현하였다. 그림 2는 본 논문에서 제안하는 블록체인의 구조이다.

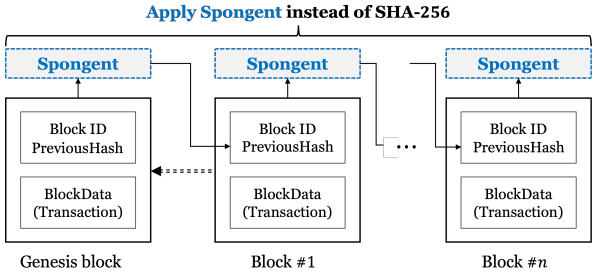

Fig. 2 Proposed blockchain architecture

해당 블록체인에서 각 블록은 BlockID, PreviousHash 그리고 BlockData로 구성되어 있다. Genesis 블록의 ID값은 0x0으로 설정하였고, 그 후 연결되는 블록의 ID 값은 0x1, 0x2와 같이 1씩 증가한다. 각 블록이 이전 블록의 해시값을 가지고 있기 때문에, 체인 형태로 연결되어 있다. 이때 사용되는 해시함수를 SHA-256 대신, 경량 해시함수인 Spongent를 사용함으로써 블록체인을 경량화 시켰다. 해시값은 BlockID, PreviousHash 그리고 BlockData를 연접하여 해시함수에 입력함으로써 계산하였다. Genesis 블록은 이전 해시값이 존재하지 않으므로 임의의 값인 '0'으로 초기화하였다. 블록 데이터는 8개의 트랜잭션으로 이루어져 있으며, 랜덤으로 생성된 값으로 설정하였다. 그림 3은 블록을 생성하는 함수이다.

```
function createBlock() {
  const blockID = "0x" + (parseInt(getLastBlock().BlockID, 16) + 1);
  const previousBlockHash = createHash(getLastBlock());
  var blockData = new Array(8);
  var txId, txData;
  for (var i = 1; i <= 8; i++){
    txId = "TxID #" + i;
    txData = crypto.randomBytes(108).toString('hex');
    blockData[i] = new tx();
    blockData[i].Id = txId;
    blockData[i].Data = txData;
  }
  const block = new Block(blockID, previousBlockHash, blockData);
  return block;
}
```

Fig. 3 Code to create block

그림 4는 임의로 3개의 블록을 생성하여 만든 블록체인 출력값이다. 블록의 ID와 이전 블록의 해시값 그리고 8개의 임의의 트랜잭션이 출력된 것을 확인할 수 있다.

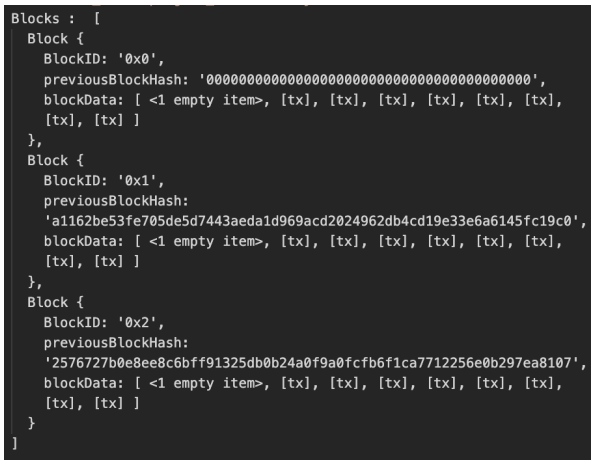

Fig. 4 Blockchain output with 3 blocks

해당 블록체인에서 사용된 Spongent 해시함수는 클래스 형태로 구현되어 있다. 또한 Javascript에서 사용되는 데이터 타입 Number의 경우 안정적으로 표현할 수 있는 정수의 최대치가 $2^{53}-1$이다. 따라서 해시함수 계산 시 블록의 ID, 이전 블록의 해시값 그리고 블록 데이터가 연접되어 입력될 때, 표현할 수 있는 정수의 최대치를 초과하여 해시값이 정상적으로 계산되지 않는다. 이와 같은 문제점을 해결하기 위해 본 논문에서는 큰 정수를 안전하게 저장하고 조작할 수 있는 BigInt를 사용하여 오버플로 없이 암호화 연산을 올바르게 수행하였다[4]. 그림 5과 6은 Spongent 암호에서 사용되는 연산인 Absorb와 Squeeze 연산을 Javascript 코드로 구현한 결과이다.

```
absorb(m, N){
    let mblocks= [];

    for (var i=0; i<N; i++){
        mblocks.push(m & BigInt('0b'+'1'.repeat(this.r)));
        m= BigInt((m/BigInt((2**this.r))));  //m = m >> this.r;
    }

    let s =0;

    for (var i =mblocks.length-1; i>=0; i--)
    {
        s = BigInt(s) ^ BigInt(this.reverse_block(mblocks[i]));
        s = this.P(s);
    }
    return s
}
```

Fig. 5 Implementation of Absorb function through Javascript

```
squeeze(s){
    let result = 0x0
    for(var i=0;i<this.n/this.r -1; i++){
        result = BigInt(result) |
        BigInt(this.reverse_block(s & BigInt('0b'+'1'.repeat(this.r))));
        result = result << BigInt(this.r);
        s = this.P(s);
    }
    result = BigInt(result) |
    BigInt(this.reverse_block(s & BigInt('0b'+'1'.repeat(this.r))));

    return result;
}
```

Fig. 6 Implementation of Squeeze function through Javascript

IV. 결론

본 논문에서는 블록체인을 사물인터넷 상에 적용시키기 위해 블록체인을 경량화 시켰다. 제안된 기법은 블록체인에서 일반적으로 사용되는 해시함수 SHA-256을 경량 해시함수인 Spongent를 사용하여 블록체인을 경량화 시키는 기법이다.

향후 코드를 최적화하고 SHA-256과 속도 및 사용되는 리소스를 비교하여 성능평가를 진행할 예정이다.

V. Acknowledgment

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No.2022-0-00627, Development of Lightweight BIoT technology for Highly Constrained Devices, 100%).

[참고문헌]

[1] Bogdanov, Andrey, et al. "SPONGENT: A lightweight hash function." International workshop on cryptographic hardware and embedded systems. Springer, Berlin, Heidelberg, 2011.

[2] Jensen, Simon Holm, Anders Møller, and Peter Thiemann. "Type analysis for JavaScript." International Static Analysis Symposium. Springer, Berlin,

Heidelberg, 2009.

[3] Schroepfer, Axel, and Florian Kerschbaum. "Secure computation in JavaScript." Proceedings of the 18th ACM conference on Computer and communications security. 2011.