JKIICE Journal of the Korea Institute of Information and Communication Engineering

한국정보통신학회논문지 Vol. 26, No. 5: 731~742, May. 2022

웹상에서의 PIPO 경량 블록암호 구현 및 성능 평가

임세진[1] · 김원웅[1] · 강예준[1] · 서화정[2*]

Implementation and performance evaluation of PIPO lightweight block ciphers on the web

Se-Jin Lim[1] · Won-Woong Kim[1] · Yea-Jun Kang[1] · Hwa-Jeong Seo[2*]

[1]Graduate Student, Division of IT Convergence Engineering, Hansung University, Seoul, 02876 Korea
[2*]Associate Professor, Division of IT Convergence Engineering, Hansung University, Seoul, 02876 Korea

요 약

PIPO는 ICISC'20에서 발표된 최신 국산 경량 블록암호로, 리소스가 제한된 IoT 상에서 구현하기 용이하도록 경량화 되어있다는 특징이 있다. 본 논문에서는 자바스크립트(Javascript), 웹어셈블리(WebAssembly)와 같은 웹 기반 언어를 사용하여 PIPO 64/128비트, 64/256비트를 구현하였다. PIPO의 비트슬라이스(BitSlice)와 TLU를 구현하여 성능평가를 진행하였으며 for문을 사용하여 작성한 일반 루프(Looped)와 for문을 풀어 작성한 루프 풀기(Unrolled)도 구현하여 성능을 비교했다. Google Chrome, Mozilla Firefox, Opera, Microsoft Edge와 같은 다양한 웹 브라우저와 윈도우즈, Linux, Mac, iOS, 안드로이드와 같은 OS별 다양한 환경에서 성능평가를 수행한다. 또한 C언어로 구현된 PIPO와 성능 비교도 수행하였다. 이는 웹상에서의 PIPO 블록암호 적용을 위한 지표로 사용될 수 있다.

ABSTRACT

PIPO is the latest domestic lightweight block cipher announced in ICISC'20, which is characterized by being lightweight to facilitate implementation on IoT with limited resources. In this paper, PIPO 64/128-bit and 64/256-bit were implemented using web-based languages such as Javascript and WebAsembly. Two methods of performance evaluation were conducted by implementing bitsice and TLU, and the performance was compared by implementing Looped written using for statements and Unrolled written for statements. It performs performance evaluations in various web browsers such as Google Chrome, Mozilla Firefox, Opera, and Microsoft Edge, as well as OS-specific environments such as Windows, Linux, Mac, iOS, and Android. In addition, a performance comparison was performed with PIPO implemented in C language. This can be used as an indicator for applying PIPO block cipher on the web.

키워드 : PIPO, 경량 블록암호, 웹구현, 자바스크립트, 웹어셈블리

Keywords : PIPO, Lightweight block cipher, Implemented on the web, Javascript, WebAssembly

Received 17 March 2022, **Revised** 1 April 2022, **Accepted** 12 April 2022

*** Corresponding Author** Hwa-Jeong Seo(E-mail:hwajeong84@gmail.com, Tel:+82-2-760-8033)
Associate Professor, Division of IT Convergence Engineering, Hansung University, Seoul, 02876 Korea

Open Access http://doi.org/10.6109/jkiice.2022.26.5.731 print ISSN: 2234-4772 online ISSN: 2288-4165

Ⅰ. 서 론

클라우드 컴퓨팅의 발전과 함께 웹상에서의 데이터 보호에 대한 중요성이 대두되고 있다. 자바스크립트(Javascript)는 운영체제나 플랫폼과 독립적으로 작동하며 확장성 또한 높아 웹 브라우저에서 주로 사용된다. 하지만 소스코드가 외부로 공개되는 HTML 언어와 함께 작성되는 경우가 많기 때문에 보안적인 측면에서 취약점이 존재한다[1]. 또한 클라이언트에서 서버로 데이터를 전송할 때 서버에서만 데이터 암·복호화를 수행하는 경우에도 데이터 노출에 대한 보안 취약점이 있다. 예를 들어 ID나 패스워드 및 개인정보 등 민감한 정보를 암호화하지 않고 평문으로 송·수신할 경우, 인가되지 않은 사용자에게 정보가 노출될 수 있는 위험이 존재한다[2]. 따라서 데이터 보호를 위해 서버에서뿐만 아니라 클라이언트에서의 암호화도 반드시 필요하다. 본 논문에서는 최신 경량 블록암호인 PIPO를 웹 기반 언어인 자바스크립트와 웹어셈블리(WebAssembly)로 구현하여 웹상에서 PIPO 알고리즘을 통한 데이터 암·복호화가 가능함을 보인다. 또한 다양한 브라우저와 OS 환경에서 구현한 알고리즘의 성능평가를 수행한다.

Ⅱ. 본 론

2.1. PIPO 블록암호

PIPO는 ICISC'20에서 제안된 국산 경량 블록암호로, 64비트의 평문을 128비트와 256비트의 키를 사용하여 암호화하는 알고리즘이다[3]. PIPO는 저전력 및 경량화를 요구하는 컴퓨팅 환경에서 기밀성을 제공하기 위해 제안된 암호 기법으로, 리소스가 제한된 IoT 상에서 구현하기 용이하도록 경량화 되어있다. 비트슬라이스(BitSlice) 기법이 적용되어 효율적인 S-box 연산이 가능하다는 특징이 있다. 한 개 블록에 대한 비트슬라이스 구현이 가능하여 높은 성능으로 소프트웨어 구현이 가능하다[4]. 또한 PIPO는 SPN(Substitution-Permutation Network) 구조를 채택하였으며, 2가지 암호화 모드를 제공한다. 표 1과 같이 128비트 키는 13번의 라운드를, 256비트 키는 17번의 라운드를 제공한다.

Table. 1 Parameters of PIPO[5]

Type	Block size	Key size	Round
64/128	64-bit	128-bit	13
64/256	64-bit	256-bit	17

아래 그림 1과 같이 PIPO의 각 라운드는 라운드 키를 포함시키는 AddRoundKey, 비선형 치환 연산을 사용하는 Substitution layer (S-Layer), 선형 회전 연산을 사용하는 Permutation layer (R-Layer)로 구성된다. S-Layer는 Lookup table을 사용하는 TLU와 효율적인 연산을 제공하는 비트슬라이스 방식으로 구현되어 있다. 비트슬라이스 방식은 11개의 비선형 비트 연산과 23개의 선형 비트 연산만 포함하여 효율적으로 활용할 수 있도록 설계되어있다[6-8]. R-Layer는 Rotation 연산을 행 단위로 수행하는데, 첫번째 행을 제외한 두번째 행부터 7, 4, 3, 6, 5, 1, 2비트 Rotation 연산을 수행한다[9].

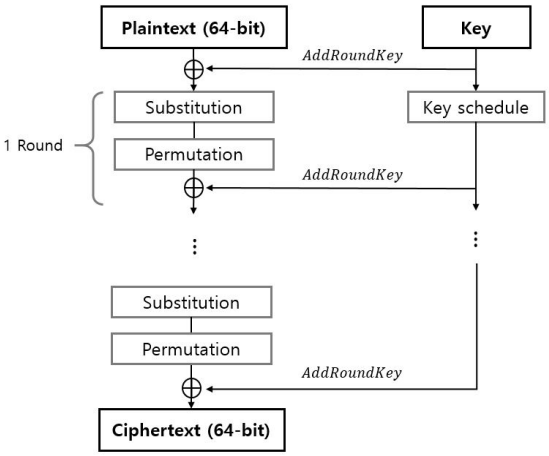

Fig. 1 Encryption structure of PIPO[3,10]

2.2. 웹어셈블리 (wasm)

웹어셈블리는 웹 브라우저에서 사용할 수 있는 효율적인 저수준 바이트 코드이다. C, C++ 등을 사용하여 작성된 프로그램은 wasm을 사용하면 자바스크립트로 컴파일이 가능하다. 자바스크립트를 로딩하기 위해서 브라우저는 텍스트로 된 .js 파일을 모두 읽어야하지만, 웹어셈블리의 경우, 사전에 최적화를 거쳐 컴파일된 wasm 파일만 인터넷을 통해 전송하므로 브라우저의 로딩 속도가 비교적 빠르다. 또한 대부분의 처리 단계가 컴파일 중에 완료되기 때문에 모바일 장치에서 배터리 소비량

을 상당히 줄일 수 있다. 웹어셈블리를 사용하는 방법 중 하나는 자바스크립트 컴파일러의 LLVM(Low-Level Virtual Machine)인 Emscripten을 사용하는 것이다. Emscripten은 C, C++ 코드를 입력하면 LLVM을 통해 생성된 바이트 코드를 자바스크립트로 변환한다. 이때 변환된 자바스크립트 코드를 자바스크립트의 하위 집합인 asm.js라고 한다. 컴파일된 asm.js 코드가 일부 렌더링을 수행 중이라면 WebGL에 의해 처리될 가능성이 높다. 이러한 방식으로 전체 파이프라인은 기술적으로 자바스크립트를 사용하고 있다. 결론적으로 웹어셈블리는 소스코드를 최적화하여 자바스크립트보다 높은 성능을 보여준다[11-12]. 그림 2는 웹어셈블리를 사용하는 과정을 도식화하여 나타낸 것이다.

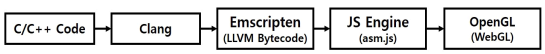

Fig. 2 Computation process of WebAssembly[11]

Ⅲ. 구 현

각 구현 환경마다 비트슬라이스, TLU 방식으로 PIPO 64/128비트, 64/256비트를 구현하였다. 추가적으로 for문을 사용하여 작성한 일반 루프(Looped)와 for문을 풀어 작성한 루프 풀기(Unrolled)도 구현하였다. 자바스크립트는 HTML의 동적인 처리를 담당하는 언어이기 때문에 PIPO의 동작 결과를 확인하기 위해서는 간단한 형태의 웹페이지가 반드시 요구된다.

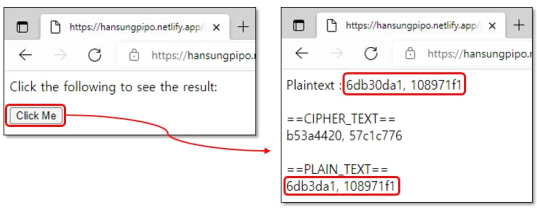

Fig. 3 PIPO implemented in Javascript

그림 3은 자바스크립트 환경에서 PIPO 암호화가 수행되는 모습이다. 메인 화면에 Button을 배치하여 Onclink 이벤트가 발생하였을 때 이벤트 리스너로 PIPO가 동작하도록 구현하였다. 그림 4는 웹어셈블리 환경에서 PIPO 암호화가 수행되는 모습이다.

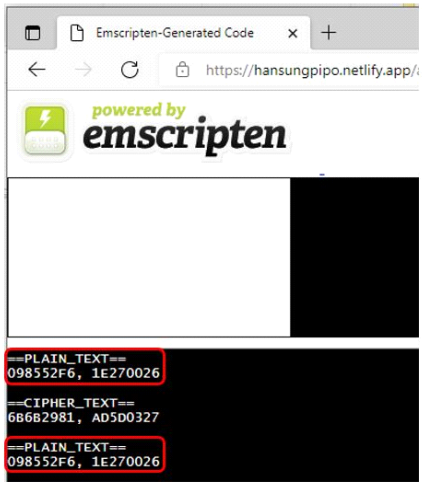

Fig. 4 PIPO implemented in WebAssembly

Ⅳ. 성능 평가

4.1. 자바스크립트 / 웹어셈블리 OS별 성능평가

성능 측정은 OS별로 각 웹 브라우저상에서 키 생성과 암호화 성능을 측정하는 방식으로 진행하였다. 복호화는 단순히 암호화의 역순으로 진행되기 때문에 암호화와 동일한 수치가 관측되므로 별도로 측정을 진행하지 않았다. 측정 단위는 각 바이트 당 클록 사이클 횟수인 cpb(cycle per byte)로 측정하였다. cpb는 암호학에서 실제 성능을 나타내는 지표로 사용되는 단위로써, 처리되는 데이터의 바이트 당 마이크로프로세서가 수행할 클록 사이클 수를 나타내는 측정 단위이다. 성능 측정은 윈도우즈, Linux, Mac, 안드로이드, iOS상에서 진행되었으며, 웹 브라우저는 Chrome, Firefox, Opera, Edge, Safari, Internet Explorer, Samsung Internet을 각 운영체제에 맞게 사용하였다. 각 OS와 브라우저별 사양은 부록의 표 2, 3과 같다. 각 웹 브라우저마다 키길이(128비트 / 256비트), 언어(자바스크립트 / 웹어셈블리), S-Layer (TLU / 비트슬라이스), 모드(일반 루프 / 루프 풀기)의 경우로 나누어서 측정하였다. 측정 결과는 부록의 표 4부터 표 8에 상세히 적혀 있다. 측정 결과에 대한 분석은 모바일과 데스크탑으로 나누어서 진행한다. 모바일에서의 측정결과는 키 생성 및 암호화에 대하여 각 OS별로 나누어서 살펴보고, 데스크탑에서의 측정결과는 키

생성과 암호화에 대하여 언어, S-Layer, 모드별로 나누어서 살펴본다.

일반적으로 자바스크립트에 비해 웹어셈블리의 성능이, TLU에 비해서는 비트슬라이스의 성능이, 일반 루프에 비해서는 루프 풀기의 성능이 더 높게 나타난다. 웹어셈블리는 C나 C++의 코드를 최적화하여 자바스크립트로 컴파일한 후 사용하므로 기존의 자바스크립트에 비해 비교적 높은 성능을 보여주며, 비트슬라이스는 입력 평문을 비트 단위로 쪼개서 마치 하드웨어와 같은 방식으로 암호 알고리즘을 동작시키는 구현 기법으로써, 한 번에 여러 개의 평문을 동시에 처리하므로[9,13,14] TLU에 비해 월등히 높은 성능을 보여준다. 또한 루프 풀기 코드의 경우 일반 루프 코드와는 달리 반복문의 조건을 확인하는 과정이 존재하지 않으므로 상대적으로 우수한 성능을 보여준다. 하지만 전반적으로 위와 같은 개념이 통용되나, 실제로 측정을 진행하였을 때 모든 경우에 적용되는 것은 아니었으며, 각 브라우저마다 서로 다른 특징이 존재함을 알 수 있었다.

4.1.1. 모바일 (iOS, 안드로이드)

우선 키 생성에 있어서 iOS는 자바스크립트에 비해 웹어셈블리가 성능이 향상된 반면에, 안드로이드는 자바스크립트에 비해 웹어셈블리의 성능이 저하됨을 알 수 있었다. 또한 루프 풀기의 경우, 안드로이드는 웹어셈블리에서만 성능이 향상되었지만, iOS는 웹어셈블리뿐만 아니라 자바스크립트 또한 성능이 향상되는 것을 알 수 있었다. 그 중에서도 웹어셈블리의 성능이 더 크게 향상되었다.

다음으로 TLU로 구현된 암호화에 있어서 안드로이드와 iOS 모두 자바스크립트에 비해 웹어셈블리의 성능이 상당히 저하되었다. 그 중 특히 안드로이드의 Firefox에서는 웹어셈블리의 성능이 측정되지 않는 현상을 보였다. 루프 풀기의 경우에는 일반 루프에 비해 전체적으로 성능이 향상되며, 특히 iOS에서는 루프 풀기로 인해 자바스크립트의 성능이 크게 향상되었다.

마지막으로 비트슬라이스로 구현된 암호화에 있어서는 웹어셈블리나 루프 풀기로 인해서 성능이 향상되지 않거나 오히려 저하되는 것을 알 수 있었다. 이는 기본적으로 비트슬라이스로 인하여 최적화가 되어있는 상태에서, 루프 풀기가 적용된 경우 임시 객체의 사용량이 늘어남에 따라 레지스터의 사용량이 증가하고, 캐시 미스(cache miss) 또한 발생하여 자바스크립트의 성능이 저하되는 것으로 보인다. 또한 자바스크립트의 가상 머신에서의 동작이 루프 풀기로 인해 코드가 길어짐에 따라 추가적인 오버헤드가 발생한 것으로 보인다 [11,15,16].

4.1.2. 데스크탑 (윈도우즈, Linux, Mac)

데스크탑의 경우, 모드와 S-Layer 측면에서는 모든 브라우저가 공통적인 경향성을 보였다. 우선 모드의 관점에서 봤을 때, 키 생성에 있어서 자바스크립트의 루프 풀기는 일반 루프에 비해 성능이 저하되었지만, TLU로 구현된 암호화에 있어서는 구현된 언어에 상관없이 루프 풀기가 일반 루프보다 성능이 높았다. 또한 비트슬라이스로 구현된 암호화의 경우, 모바일과 마찬가지로 성능이 향상되지 않거나 오히려 성능이 저하되었다. TLU와 비트슬라이스로 구현된 암호화에서는 공통적으로 루프 풀기에 비해 일반 루프의 성능이 향상되었음을 알 수 있다. 이는 모바일에서와 마찬가지로 루프 풀기에 의한 레지스터 사용량의 증가와 cache miss, 그리고 긴 코드로 인한 오버헤드에 의한 것으로 보인다. 다음으로 S-Layer의 관점에서 보았을 때, 모든 데스크탑 환경에서 TLU에 비해 비트슬라이스의 성능이 눈에 띄게 향상되었다. 마지막으로 언어의 관점에서 보았을 때에는 각 브라우저별로 서로 다른 특징을 지니고 있기 때문에 다음 절에서 브라우저별로 나누어 설명한다.

4.1.2.1. Chrome

키 생성의 경우, 256비트 키를 사용하는 루프 풀기에 대해 웹어셈블리의 성능이 향상되는 것을 알 수 있었다. 그러나 Mac OS 환경에서는 128비트 키를 사용하는 일반 루프의 경우, 웹어셈블리의 성능이 저하되었다. TLU로 구현된 암호화의 경우, Linux와 Mac의 루프 풀기를 제외하고 웹어셈블리의 성능이 저하되는 것이 관측되었다. 또한 비트슬라이스로 구현된 암호화의 경우에는 전반적으로 웹어셈블리에서 성능이 향상되었으며, 그 중에서도 리눅스 환경에서 성능이 가장 크게 향상되었다. 그러나 윈도우즈의 경우에는 성능이 크게 다르지 않았다.

4.1.2.2. Firefox

Firefox에서 웹어셈블리의 성능 향상이 가장 크게 나타났다. 하지만 기본적으로 다른 브라우저에 비해 Firefox에

서의 자바스크립트 성능이 현저히 낮기 때문에 성능이 크게 향상되었음에도 불구하고 여전히 비교적 낮은 성능을 보인다. 예외적으로 Mac에서 Firefox의 웹어셈블리가 다른 브라우저에 비해 높은 성능을 보여주기도 한다.

4.1.2.3. Safari

Safari의 경우, 다른 브라우저들에 비해 예외사항 없이 모든 경우에 웹어셈블리에서 성능이 향상되었다.

4.1.2.4. Opera

키 생성과 TLU로 구현된 암호화의 경우, 윈도우즈의 128비트 일반 루프와 Mac의 일반 루프를 제외한 모든 경우에서 웹어셈블리의 성능이 향상되었다. 또한 비트슬라이스로 구현된 암호화의 경우에는 윈도우즈를 제외한 모든 경우에 성능이 향상되는 것을 알 수 있었다.

4.1.2.5. Internet Explorer

Internet Explorer의 경우 웹어셈블리로 구현된 웹페이지를 지원하지 않으므로 성능 측정 대상에서 제외하였다.

4.1.2.6. Edge

Edge는 키 생성과 TLU로 구현된 암호화에 있어서 일반 루프의 경우 자바스크립트가, 루프 풀기의 경우 웹어셈블리가 더 좋은 성능을 보였다. 하지만 비트슬라이스로 구현된 암호화에서는 윈도우즈의 일반 루프를 제외하고는 웹어셈블리에 의해 성능이 향상되었다.

4.2. C언어 대비 OS별 성능평가

데스크탑 환경에서 각 브라우저에 대해 웹 기반 언어인 자바스크립트와 웹어셈블리로 구현된 PIPO의 성능을 측정하였으며, C언어로 구현된 PIPO와 비교하였다. 또한 성능 측정은 C언어 대비 cpb 성능을 %로 나타내어 표기하였다. 각 OS별 그래프는 키길이와 구현 언어, 모드를 구분하지 않고 브라우저별 성능을 측정한 결과이다. 가장 높은 성능을 보인 브라우저와 가장 낮은 성능을 보인 브라우저를 best와 worst로 나타내었다. 추가적으로 구현 언어와 모드를 구분하여 상세히 설명한다.

4.2.1. 윈도우즈

그림 5는 윈도우즈에서의 성능평가를 나타낸 그래프이다. Opera와 Edge에서 비교적 높은 성능을, Chrome, Firefox, Internet Explorer에서 비교적 낮은 성능을 보이고 있다.

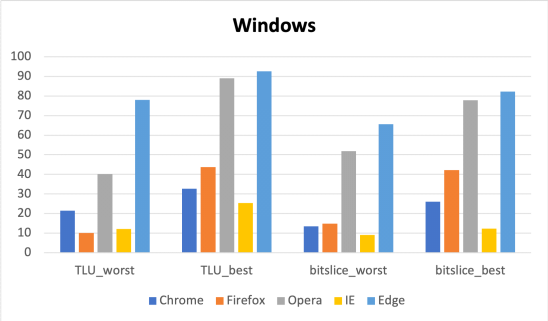

Fig. 5 Performance evaluation by browser compared to C language in Windows

일반 루프 자바스크립트의 경우, TLU상에서 18.6%-92.6%의 성능 차이를 보였다. 그 중 Firefox가 18.6%로 가장 낮았으며, Edge가 92.6%로 가장 높았다. 또한 비트슬라이스 상에서는 9.1%-82.2%의 성능 차이를 보였다. 그 중 Internet Explorer가 9.1%로 가장 낮았으며, Edge가 82.2%로 가장 높았다. 일반 루프 웹어셈블리의 경우, TLU상에서 36.3%-79.0%의 성능 차이를 보였다. 그 중 Firefox가 36.3%로 가장 낮았으며, Edge가 79.0%로 가장 높았다. 또한 비트슬라이스 상에서는 34.7%-75.2%의 성능 차이를 보였다. 그 중 Firefox가 34.7%로 가장 낮았으며, Opera가 75.2%로 가장 높았다. 보통 128비트와 256비트의 성능 차이가 1-3%가 차이나는 반면, 일반 루프 웹어셈블리에서의 Opera의 경우 128비트와 256비트가 최대 38.7%까지 차이가 나는 것을 확인하였다. 루프 풀기 자바스크립트의 경우, TLU상에서 10.0%-87.1%의 성능 차이를 보였다. 그 중 Firefox가 10.0%로 가장 낮았으며, Edge가 87.1%로 가장 높았다. 또한 비트슬라이스 상에서는 12.1%-66.9%의 성능 차이를 보였다. 그 중 Internet Explorer가 12.1%로 가장 낮았으며, Edge가 66.9%로 가장 높았다. 루프 풀기 웹어셈블리의 경우, TLU상에서 42.7%-89.5%의 성능 차이를 보였다. 그 중 Chrome이 42.7%로 가장 낮았으며, Edge가 89.5%로 가장 높았다. 비트슬라이스 상에서는 36.7%-77.8%의 성능 차이를 보였다. 그 중 Chrome이 36.7%로 가장 낮았으며, Opera가 77.8%로 가장 높았다. 윈도우즈의 경우 best / worst의 차이가 가장 컸으며 웹어셈블리에서는

worst임에도 불구하고 최소 34.7% 이상의 C대비 성능을 보여주었다.

4.2.2. Linux

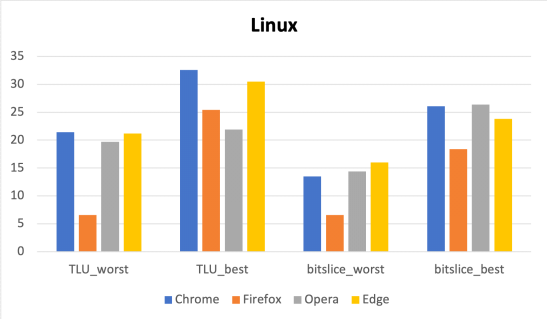

Fig. 6 Performance evaluation by browser compared to C language in Linux

그림 6은 Linux에서의 성능평가를 나타낸 그래프이다. Firefox에서만 비교적 낮은 성능을 보이고 있으며, Firefox를 제외한 브라우저에서는 유사한 성능을 보이고 있다.

일반 루프 자바스크립트의 경우, TLU상에서 15.3%-32.6%의 성능 차이를 보였다. 그 중 Firefox가 15.3%로 가장 낮았으며, Chrome이 32.6%로 가장 높았다. 또한 비트슬라이스 상에서는 8.6%-18.7%의 성능 차이를 보였다. 그 중 Firefox가 8.6%로 가장 낮았으며, Chrome이 18.7%로 가장 높았다. 일반 루프 웹어셈블리의 경우, TLU상에서, 21.7%-32.9%의 성능 차이를 보였다. 그 중 Firefox가 21.7%로 가장 낮았으며, Opera가 32.9%로 가장 높았다. 또한 비트슬라이스 상에서는 14.8%-25.3%의 성능 차이를 보였다. 그 중 Firefox가 14.8%로 가장 낮았으며, Opera가 25.3%로 가장 높았다. 루프 풀기 자바스크립트의 경우, TLU상에서 6.6%-22.2%의 성능 차이를 보였다. 그 중 Firefox가 6.6%로 가장 낮았으며, Edge가 22.2%로 가장 높았다. 또한 비트슬라이스 상에서는 6.6%-16.8%의 성능 차이를 보였다. 그 중 Firefox가 6.6%로 가장 낮았으며, Chrome가 16.8%로 가장 높았다. 루프 풀기 웹어셈블리의 경우, TLU상에서 21.8%-32.1%의 성능 차이를 보였다. 그 중 Firefox이 21.8%로 가장 낮았으며, Chrome가 32.1%로 가장 높았다. 비트슬라이스 상에서는 16.5%-26.4%의 성능 차이를 보였다. 그 중 Firefox가 16.5%로 가장 낮았으며, Opera가 26.4%로 가장 높았다. Linux의 경우 약 15% 내외의 best / worst 차이를 갖고 있으며, 최대 32.6%로 윈도우즈에 비해 저조한 성능을 보여주었다.

4.2.3. Mac

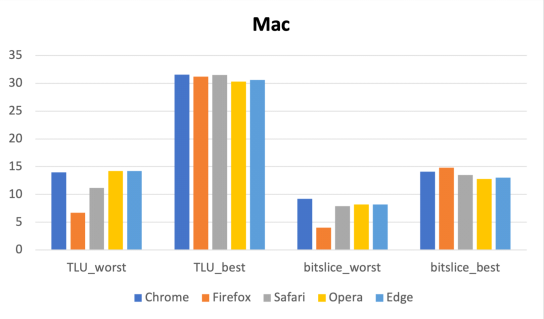

Fig. 7 Performance evaluation by browser compared to C language in Mac

그림 7은 Mac에서의 성능평가를 나타낸 그래프이다. Linux와 마찬가지로 Firefox에서만 비교적 낮은 성능을 보이고 있으며, Firefox를 제외한 브라우저에서는 유사한 성능을 보이고 있다.

일반 루프 자바스크립트의 경우, TLU상에서 7.3%-16.1%의 성능 차이를 보였다. 그 중 Firefox가 7.3%로 가장 낮았으며, Chrome이 16.1%로 가장 높았다. 또한 비트슬라이스 상에서는 5.5%-12.8%의 성능 차이를 보였다. 그 중 Firefox가 5.5%로 가장 낮았으며, Chrome이 12.8%로 가장 높았다. 일반 루프 웹어셈블리의 경우, TLU상에서, 12.6%-15.4%의 성능 차이를 보였다. 그 중 Firefox가 12.6%로 가장 낮았으며, Safari가 15.4%로 가장 높았다. 또한 비트슬라이스 상에서는 12.2%-14.8%의 성능 차이를 보였다. 그 중 Edge가 12.2%로 가장 낮았으며, Firefox가 14.8%로 가장 높았다. 루프 풀기 자바스크립트의 경우, TLU상에서 6.7%-28.2%의 성능 차이를 보였다. 그 중 Firefox가 6.7%로 가장 낮았으며, Safari가 28.2%로 가장 높았다. 또한 비트슬라이스 상에서는 4.0%-10.1%의 성능 차이를 보였다. 그 중 Firefox가 4.0%로 가장 낮았으며, Chrome가 10.1%로 가장 높았다. 루프 풀기 웹어셈블리의 경우, TLU상에서 29.4%-31.6%의 성능 차이를 보였다. 그 중 Opera가 29.4%로 가장 낮았으며, Chrome가 31.6%로 가장 높았다. 비트슬라이스 상에서는 10.9%-14.4%의 성능 차이를 보였다. 그 중 Edge가 10.9%로 가장 낮았으며, Firefox가 14.4%로 가장 높았다. Mac의 경우 전체적으

로 성능이 가장 저조하였으며, 또한 Firefox를 제외하고는 각 브라우저별 성능 차이가 거의 존재하지 않았다. 또한 각 환경별 best / worst가 계속 변동될 정도로 각 브라우저 사이의 큰 차이가 없었으며, 그에 따라 성능을 측정하여 브라우저별 성능을 비교하는 것에는 큰 의미가 없었다.

V. 응 용

그림 8은 웹페이지 상에 업로드한 이미지를 자바스크립트로 구현한 PIPO 알고리즘을 적용하여 암·복호화한 모습이다. 이미지의 크기에 관계없이 PIPO의 입력 비트와 동일하게 연산될 수 있도록 canvas의 크기를 400x400으로 고정하였다. 이미지 데이터는 하나의 픽셀에 색상과 투명도를 의미하는 RGBA 값을 포함한다. 5x5 이미지의 경우, 배열의 크기는 5x5x4=100이 된다. 즉, 하나의 픽셀은 배열 4개를 차지한다. 픽셀 한 개당 32비트의 크기를 가지고 있으므로 한번 암호화할 때 2개의 픽셀 즉, 8개의 배열이 암호화 된다. 2개의 픽셀을 기준으로 하나의 마스터키로부터 생성된 서로 다른 라운드 키를 블록 크기만큼 나누어 적용하여 암호화하였다. 따라서 각 블록에 각기 다른 암호화키가 적용되었으며, 그림 8처럼 원본 이미지를 유추할 수 없도록 암호화가 수행된 것을 알 수 있다. 암호화에 사용한 라운드 키는 2차원 배열에 저장하였다가 복호화 시에 사용하였다.

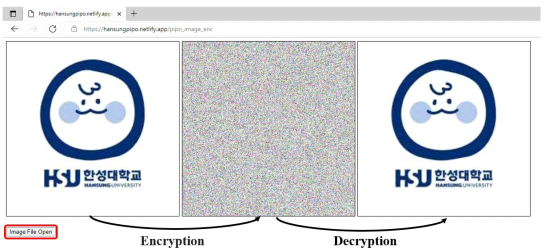

Fig. 8 Image Encryption by PIPO

프론트엔드에서 백엔드로 데이터를 주고받을 때 다른 사용자에게 데이터가 노출될 가능성이 있다. 따라서 프로트엔드에서 암호화를 하는 것은 중요한 문제이다. 또한 기밀 문서나 DB 정보와 같이 중요한 데이터가 유출될 경우 큰 피해를 초래할 수 있으며 2차 공격으로 이어질 수 있다. 이를 보완하기 위해 경량 블록암호인 PIPO를 통해 데이터를 암호화하여 송·수신하면 스니핑 공격이 이루어져도 정보를 안전하게 보호할 수 있다.

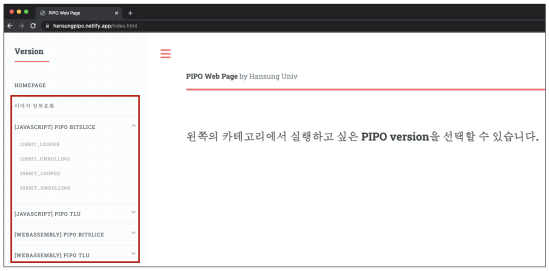

Fig. 9 Website for PIPO

https://hansungpipo.netlify.app/에서 이미지 암·복호화 및 자바스크립트, 웹어셈블리로 구현된 PIPO가 동작하는 것을 볼 수 있다. 구현한 결과물들을 하나의 웹페이지에 모아 도메인을 할당받아 배포한 웹페이지로, 그림 9와 같다. 따라서 외부에서 누구나 접속하여 암호화를 테스트해볼 수 있다. 웹기반 언어는 다른 언어와 달리 프레임워크와 같은 플랫폼에 독립적이며, 모바일 환경에서도 PIPO 암호화를 수행할 수 있다. 이처럼 홈페이지를 통해 암호화를 수행할 수 있다는 것은 큰 강점을 가진다.

VI. 결 론

본 논문에서는 경량 블록암호 PIPO의 효율적인 구현을 위해 PIPO를 자바스크립트와 웹어셈블리로 구현하여 성능을 측정하였다. Chrome, Firefox, Safari, Opera, Internet Explorer, Edge 브라우저 상에서 성능을 측정하였으며, 그 결과 각 브라우저 별로 다른 특징을 보였다. 또한 C언어로 구현된 PIPO와 성능을 비교하였다. 더 나아가 자바스크립트 환경에서 블록암호 PIPO를 통해 이미지 데이터의 암·복호화를 수행하였다. 이를 통해 PIPO가 다양한 클라우드 컴퓨팅 분야에 적용될 수 있음을 확인할 수 있었다. 추후 웹어셈블리 환경에서도 데이터 암호화를 진행할 예정이다.

부 록

Table. 2 Measurement specification by OS

OS Version	CPU	RAM	Device
Windows 10 64bit	AMD Ryzen 7 4800H powered by 2.9GHz	16GB	-
Ubuntu 20.04.2 LTS	Intel(R) core(TM) i3-6100U CPU @ 2.30GHz	16GB	-
Mac 11.3.1	Intel Core i7 by 2.6 GHz 6core	16GB	-
Android 8.0.0	4x Samsung Exynos M2 @ 2.31GHz, 8core	6GB	Galaxy Note 8
iOS 14.4.2	2.65GHz	4GB	iPhone 11

Table. 3 Measuring browser environment by OS

OS	Browser	Version
Windows	Chrome	90.0.4430.212
	Firefox	88.0.1
	Opera	76.0
	Internet Explorer	2004
	Edge	90.0.818.62
Linux	Chrome	90.0.4430.212
	Firefox	88.0.1
	Opera	76.0
	Edge	91.0.864.27 (beta)
Mac	Chrome	90.0.4430.212
	Firefox	88.0.1
	Safari	14.1 (16611.1.21.161.6)
	Opera	76.0.4017.137
	Edge	90.0.818.66
Android	Chrome	90.0.4430.210
	Firefox	88.1.3
	Opera	63.3.3216.58675
	Samsung Internet	14.0.1.62
	Edge	46.03.4.5155
iOS	Chrome	87.0.4280.163
	Firefox	33.1
	Safari	14.5.1
	Opera	3.0.4
	Internet Explorer	1.0.1
	Edge	46.3.13

Table. 4 Measurement of performance in Windows.
C: Chrome, FF: Firefox, O: Opera, IE: Internet Explorer, E: Edge

	Method	Key	Enc(TLU)	Enc(Bitslice)
Looped	128bit_JS_C	147.175	2487.475	494.45
	128bit_JS_FF	350.175	6746.125	947.575
	128bit_JS_O	105.85	2750.65	260.275
	128bit_JS_IE	403.825	4801.675	1558.75
	128bit_JS_E	59.45	1392	251.575
	256bit_JS_C	181.25	3277.725	595.225
	256bit_JS_FF	388.6	8739.15	1276.725
	256bit_JS_O	109.475	3275.55	318.275
	256bit_JS_IE	1436.95	6307.5	2014.05
	256bit_JS_E	68.875	1794.375	314.65
	128bit_asm_C	207.35	2905.8	492.275
	128bit_asm_FF	225.475	3415.475	429.2
	128bit_asm_O	203	3079.075	265.35
	128bit_asm_E	77.575	1595.725	264.625
	256bit_asm_C	206.625	3920.075	599.575
	256bit_asm_FF	222.575	4495.725	597.4
	256bit_asm_O	81.2	2110.475	337.85
	256bit_asm_E	84.1	2129.325	337.125
Unrolled	128bit_JS_C	150.8	1333.275	581.45
	128bit_JS_FF	581.45	6162.5	988.175
	128bit_JS_O	126.15	1393.45	290
	128bit_JS_IE	350.175	5230.875	1566
	128bit_JS_E	63.075	747.475	292.175
	256bit_JS_C	195.75	1715.35	676.425
	256bit_JS_FF	713.4	8167.125	1257.875
	256bit_JS_O	135.575	1804.525	368.3
	256bit_JS_IE	469.075	6178.45	1961.125
	256bit_JS_E	79.75	941.775	368.3
	128bit_asm_C	152.25	1435.5	483.575
	128bit_asm_FF	158.05	1410.125	419.05
	128bit_asm_O	52.925	717.75	256.65
	128bit_asm_E	55.1	718.475	255.2
	256bit_asm_C	154.425	1758.125	646.7
	256bit_asm_FF	169.65	1905.3	541.575
	256bit_asm_O	55.825	943.95	329.875
	256bit_asm_E	59.45	935.25	329.875

Table. 5 Measurement of performance in Linux

	Method	Key	Enc(TLU)	Enc(Bitslice)
Looped	128bit_JS_C	119.025	2746.2	581.9
	128bit_JS_FF	435.85	5635.575	1069.5
	128bit_JS_O	187.45	2877.3	588.8
	128bit_JS_E	192.625	2948.025	566.95
	256bit_JS_C	117.875	3711.05	818.8
	256bit_JS_FF	534.175	7240.975	1390.35
	256bit_JS_O	201.825	3650.1	748.075
	256bit_JS_E	208.725	3941.625	702.075
	128bit_asm_C	207.575	2712.85	369.15
	128bit_asm_FF	205.85	4029.025	679.65
	128bit_asm_O	139.725	2698.475	378.35
	128bit_asm_E	240.35	3056.7	365.7
	256bit_asm_C	219.65	3546.025	465.175
	256bit_asm_FF	338.675	5249.75	706.1
	256bit_asm_O	218.5	3600.075	478.4
	256bit_asm_E	258.175	3951.4	477.25
Unrolled	128bit_JS_C	154.675	1499.025	662.4
	128bit_JS_FF	633.075	4518.925	1124.7
	128bit_JS_O	219.65	1578.375	646.875
	128bit_JS_E	220.8	1457.625	615.825
	256bit_JS_C	185.15	1918.775	999.925
	256bit_JS_FF	1034.425	5763.225	1389.775
	256bit_JS_O	270.25	1988.35	844.1
	256bit_JS_E	234.6	1788.25	768.2
	128bit_asm_C	165.025	1010.85	361.675
	128bit_asm_FF	105.225	1293.175	642.85
	128bit_asm_O	159.275	970.025	362.25
	128bit_asm_E	216.2	955.075	362.825
	256bit_asm_C	156.4	1245.45	480.7
	256bit_asm_FF	101.2	1958.45	874
	256bit_asm_O	158.125	1291.45	474.95
	256bit_asm_E	217.925	1257.525	486.45

Table. 6 Measurement of performance in Mac

	Method	Key	Enc(TLU)	Enc(Bitslice)
Looped	128bit_JS_C	54.6	1544.4	275.6
	128bit_JS_FF	203.45	3220.1	511.55
	128bit_JS_S	132.6	2073.5	326.3
	128bit_JS_O	79.95	1578.85	286.65
	128bit_JS_E	76.7	1617.2	287.95
	256bit_JS_C	68.25	2046.85	327.6
	256bit_JS_FF	275.6	4128.15	644.15
	256bit_JS_S	144.95	2720.9	390.65
	256bit_JS_O	85.15	2042.95	348.4
	256bit_JS_E	81.25	2036.45	352.3
	128bit_asm_C	59.8	1690.65	221
	128bit_asm_FF	76.7	1746.55	200.85
	128bit_asm_S	58.5	1617.85	243.1
	128bit_asm_O	81.25	1683.5	236.6
	128bit_asm_E	80.6	1660.75	243.75
	256bit_asm_C	66.3	2227.55	295.75
	256bit_asm_FF	82.55	2461.55	259.35
	256bit_asm_S	63.05	2107.95	310.05
	256bit_asm_O	92.3	2166.45	303.55
	256bit_asm_E	87.75	2173.6	301.6
Unrolled	128bit_JS_C	59.15	767	295.1
	128bit_JS_FF	298.35	2483.65	508.95
	128bit_JS_S	80.6	625.95	330.2
	128bit_JS_O	79.3	764.4	317.85
	128bit_JS_E	78	787.15	319.15
	256bit_JS_C	71.5	993.2	379.6
	256bit_JS_FF	464.75	3227.9	624
	256bit_JS_S	103.35	962.65	409.5
	256bit_JS_O	96.85	1000.35	400.4
	256bit_JS_E	93.6	996.45	393.9
	128bit_asm_C	43.55	588.25	219.05
	128bit_asm_FF	52.65	587.6	202.8
	128bit_asm_S	34.45	599.3	248.95
	128bit_asm_O	61.75	596.7	227.5
	128bit_asm_E	58.5	592.8	239.85
	256bit_asm_C	44.2	769.6	289.25
	256bit_asm_FF	55.9	769.6	260
	256bit_asm_S	38.35	765.05	313.95
	256bit_asm_O	63.05	781.95	306.15
	256bit_asm_E	62.4	781.95	299

Table. 7 Measurement of performance in Android.
SI: Samsung Internet

	Method	Key	Enc(TLU)	Enc(Bitslice)
Looped	128bit_JS_C	302.0325	4453.68	926.31
	128bit_JS_FF	612.7275	5426.7675	1271.655
	128bit_JS_SI	279.51	4502.19	939.015
	128bit_JS_O	323.4	4509.6975	939.015
	128bit_JS_E	314.16	4397.6625	968.4675
	256bit_JS_C	358.6275	5783.6625	1081.6575
	256bit_JS_FF	686.07	6989.4825	1607.76
	256bit_JS_SI	351.6975	5828.13	1096.6725
	256bit_JS_O	360.9375	5851.23	1219.1025
	256bit_JS_E	367.8675	5744.97	1111.11
	128bit_asm_C	833.3325	7802.025	1153.845
	128bit_asm_FF	-	-	7110.18
	128bit_asm_SI	950.565	7325.5875	1157.31
	128bit_asm_O	933.24	7683.6375	1155.5775
	128bit_asm_E	483.3675	5416.3725	703.9725
	256bit_asm_C	854.1225	9908.745	1504.965
	256bit_asm_FF	-	-	9254.4375
	256bit_asm_SI	1157.8875	9566.2875	1502.0775
	256bit_asm_O	980.0175	8496.18	1499.7675
	256bit_asm_E	485.6775	7065.7125	915.915
Unrolled	128bit_JS_C	308.9625	2394.8925	1005.4275
	128bit_JS_FF	796.95	3326.4	1271.655
	128bit_JS_SI	310.1175	2307.69	967.89
	128bit_JS_O	318.78	2430.6975	1047.0075
	128bit_JS_E	314.7375	2256.2925	985.7925
	256bit_JS_C	384.0375	3045.735	1256.0625
	256bit_JS_FF	1119.195	4284.4725	1622.775
	256bit_JS_SI	373.065	2931.9675	1106.49
	256bit_JS_O	388.08	3005.8875	1226.61
	256bit_JS_E	384.0375	2784.1275	1137.675
	128bit_asm_C	683.1825	2805.495	1127.28
	128bit_asm_FF	645.0675	35423.85	7012.005
	128bit_asm_SI	912.45	2534.07	1133.055
	128bit_asm_O	836.7975	2811.27	1125.5475
	128bit_asm_E	370.755	2308.845	708.5925
	256bit_asm_C	719.565	3669.435	1453.5675
	256bit_asm_FF	-	-	9121.035
	256bit_asm_SI	984.6375	3322.3575	1465.695
	256bit_asm_O	812.5425	3663.66	1459.92
	256bit_asm_E	340.725	3003	902.055

Table. 8 Measurement of performance in iOS
S: Safari

	Method	Key	Enc(TLU)	Enc(Bitslice)
Looped	128bit_JS_C	194.775	1878.85	284.2125
	128bit_JS_FF	203.3875	1867.5875	284.875
	128bit_JS_S	198.75	1851.025	280.2375
	128bit_JS_O	190.1375	1855	281.5625
	128bit_JS_IE	3203.1875	26880.9375	6140.05
	128bit_JS_E	212	1861.625	282.225
	256bit_JS_C	227.2375	2443.9625	347.8125
	256bit_JS_FF	231.2125	2408.85	352.45
	256bit_JS_S	211.3375	2391.625	343.175
	256bit_JS_O	217.3	2390.9625	343.8375
	256bit_JS_IE	4008.125	37011.8875	7955.3
	256bit_JS_E	229.225	2400.2375	345.1625
	128bit_asm_C	161.65	2952.1	443.2125
	128bit_asm_FF	129.1875	2932.8875	437.9125
	128bit_asm_S	170.2625	2951.4375	446.525
	128bit_asm_O	158.3375	2948.7875	443.2125
	128bit_asm_E	150.3875	2944.8125	441.8875
	256bit_asm_C	172.25	3836.5375	562.4625
	256bit_asm_FF	137.8	3834.55	561.8
	256bit_asm_S	187.4875	3847.1375	565.1125
	256bit_asm_O	187.4875	3847.1375	566.4375
	256bit_asm_E	173.575	3836.5375	565.1125
	128bit_JS_C	170.2625	667.8	305.4125
Unrolled	128bit_JS_FF	180.8625	657.8625	307.4
	128bit_JS_S	178.2125	645.275	302.7625
	128bit_JS_O	170.925	645.9375	304.0875
	128bit_JS_IE	2950.775	20841.5875	6098.3125
	128bit_JS_E	188.8125	649.9125	306.075
	256bit_JS_C	200.7375	828.7875	368.35
	256bit_JS_FF	206.7	820.8375	376.3
	256bit_JS_S	206.7	807.5875	365.7
	256bit_JS_O	212.6625	809.575	366.3625
	256bit_JS_IE	3523.8375	26839.2	7894.35
	256bit_JS_E	213.325	816.2	367.6875
	128bit_asm_C	80.825	1235.5625	461.7625
	128bit_asm_FF	74.2	1223.6375	441.225
	128bit_asm_S	102.6875	1248.15	465.7375
	128bit_asm_O	107.325	1250.1375	457.125
	128bit_asm_E	98.7125	1248.15	460.4375
	256bit_asm_C	88.775	1592.65	584.9875
	256bit_asm_FF	80.1625	1581.3875	558.4875
	256bit_asm_S	111.9625	1600.6	580.35
	256bit_asm_O	111.3	1607.8875	581.675
	256bit_asm_E	117.925	1603.9125	578.3625

ACKNOWLEDGEMENT

This work was partly supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIT) (No.2018-0-00264, Research on Blockchain Security Technology for IoT Services, 30%) and this work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2021-0-00540, Development of Fast Design and Implementation of Cryptographic Algorithms based on GPU/ASIC, 30%) and this work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. NRF-2020R1F1A 1048478, 30%). This research was financially supported by Hansung University for Hwajeong Seo.

REFERENCES

[1] S. -C. Kang, J. -S. Park, "Security issues in HTML5's next generation web standard environment," *Review of KIISC*, vol. 24, no. 4, pp. 44-55, Aug. 2014

[2] S. J. Yoon, J. H. Jung, and H. K. Kim, "A study on JavaScript-based attack techniques using HTML5," *Journal of The Korea Institute of Information Security and Cryptology*, vol. 25, no. 5, pp. 74-80, Oct. 2015

[3] H. G. Kim, Y. J. Jeon, G. Y. Kim, J. S. Kim, B. -Y. Sim, D. -G. Han, H. J. Seo, S. G. Kim, S. H. Hong, J. C. Sung, and D. J. Hong, "PIPO: A Lightweight Block Cipher with Efficient Higher-Order Masking Software Implementations," in *International Conference on Information Security and Cryptology*, Seoul, South Korea, vol. 12593, pp. 99-122, 2020

[4] J. S. Kim, S. G. Kim, S. Y. Kim, D. J. Hong, J. C. Sung, and S. H. Hong, "MILP-Aided Division Property and Integral Attack on Lightweight Block Cipher PIPO," *Journal of the Korea Institute of Information Security & Cryptology*, vol. 31, no. 5, pp. 875-888, Oct. 2021

[5] S. W. Eum, H. D. Kwon, H. J. Kim, K. B. Jang, H. J. Kim, J. H. Park, G. J. Song, M. J. Sim, and H. J. Seo, "Optimized Implementation of Block Cipher PIPO in Parallel-Way on 64-bit ARM Processors," *KIPS Transactions on Computer and Communication Systems*, vol. 10, no. 8, pp. 223-230, Aug. 2021

[6] Y. J. Kwak, Y. B. Kim, and S. C. Seo, "Benchmarking Korean Block Ciphers on 32-Bit RISC-V Processor," *Journal of the Korea Institute of Information Security & Cryptology*, vol. 31, no. 3, pp. 331-340, Jun. 2021

[7] J. G. Song, Y. B. Kim, and S. C. Seo, "High-Speed Fault Attack Resistant Implementation of PIPO Block Cipher on ARM Cortex-A," *Institute of Electrical and Electronics Engineers*, vol. 9, pp. 162893-162908, Dec. 2021

[8] H. J. Kim, M. J. Sim, S. W. Eum, K. B. Jang, G. J. Song, H. J. Kim, H. D. Kwon, W. -K. Lee, and H. J. Seo, "Masked Implementation of PIPO Block Cipher on 8-bit AVR Microcontrollers," in *Information Security Applications: 22nd International Conference*, Jeju Island, South Korea, vol. 13009, pp. 171-182, 2021

[9] I. Y. Kim, B. J. Seok, and C. H. Lee, "A Study of Fast Implementation of Korea Block Ciphers PIPO, HIGHT, and CHAM," *Journal of Digital Contents Society*, vol. 22, no. 12, pp. 2063-2075, Dec. 2021

[10] K. B. Jang, G. J. Song, H. D. Kwon, S. W. Uhm, H. J. Kim, W. -K. Lee, and H. J. Seo, "Grover on PIPO," *Electronics 2021*, vol. 10, no. 10, May. 2021

[11] H. J. Seo and H. W. Kim, "Low-power encryption algorithm block cipher in Javascript," *Journal of information and communication convergence engineering*, vol. 12, no. 4, pp. 252-256, Dec. 2014

[12] A. Haas, A. Rossberg, D. L. Schuff, B. L. Titzer, M. Holman, D. Gohman, L. Wagner, A. Zakai, and J. Bastien, "Bringing the web up to speed with WebAssembly," in *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, Barcelona, Spain, pp. 185-200, 2017

[13] C. Rebeiro, D. Selvakumar, and A. S. L. Devi, "Bitslice Implementation of AES," in *International Conference on Cryptology and Network Security*, Suzhou, China, vol. 4301, pp. 203-212, 2006

[14] W. Zhang, Z. Bao1, D. Lin, V. Rijmen, B. Yang, and I. Verbauwhede, "RECTANGLE: A Bit-slice Lightweight Block Cipher Suitable for Multiple Platforms," *Science China Information Sciences*, vol. 58, pp. 1-15, Nov. 2015

[15] E. Stark, M. Hamburg, and D. Boneh, "Symmetric Cryptography in Javascript," in *2009 Annual Computer Security Applications Conference*, Honolulu: HI, USA,pp.

373-381, 2009

[16] C. H. Park, T. H. Park, H. J. Seo, and H. W. Kim, “Optimization of CHAM Encryption Algorithm Based on Javascript,” in *2018 Tenth International Conference on Ubiquitous and Future Networks*, Prague, Czech Republic, pp. 774-778, 2018

임세진(Se-Jin Lim)

2022년 2월 : 한성대학교 컴퓨터공학부 졸업
2022년 3월~현재 : 한성대학교 IT융합공학부 석사과정
※관심분야 : 인공지능 보안, 정보보안

김원웅(Won-Woong Kim)

2022년 2월 : 한성대학교 컴퓨터공학부 졸업
2022년 3월~현재 : 한성대학교 IT융합공학부 석사과정
※관심분야 : 인공지능, 블록체인

강예준(Yea-Jun Kang)

2022년 2월 : 한성대학교 컴퓨터공학부 졸업
2022년 3월~현재 : 한성대학교 IT융합공학부 석사과정
※관심분야 : 블록체인, 인공지능 보안

서화정(Hwa-Jeong Seo)

2010년 2월: 부산대학교 컴퓨터공학과 졸업
2012년 2월 : 부산대학교 컴퓨터공학과 석사
2015년 4월~5월 : 싱가포르 난양공대 인턴쉽
2016년 2월 : 부산대학교 컴퓨터공학과 박사
2017년 3월 : 싱가포르 과학기술청 연구원
2017년 4월~현재 : 한성대학교 조교수
※관심분야 : 암호구현