Article

Efficient Implementation of NIST LWC ESTATE Algorithm Using OpenCL and Web Assembly for Secure Communication in Edge Computing Environment

BoSun Park and Seog Chung Seo *

Department of Financial Information Security, Kookmin University, Seoul 02707, Korea; 20175206@kookmin.ac.kr
* Correspondence: scseo@kookmin.ac.kr; Tel.: +82-02-910-4742

Abstract: In edge computing service, edge devices collect data from a number of embedded devices, like sensors, CCTVs (Closed-circuit Television), and so on, and communicate with application servers. Since a large portion of communication in edge computing services are conducted in wireless, the transmitted data needs to be properly encrypted. Furthermore, the application servers (resp. edge devices) are responsible for encrypting or decrypting a large amount of data from edge devices (resp. terminal devices), the cryptographic operation needs to be optimized on both server side and edge device side. Actually, the confidentiality and integrity of data are essential for secure communication. In this paper, we present two versions of security software which can be used on edge device side and server side for secure communication between them in edge computing environment. Our softwares are basically web-based application because of its universality where the softwares can be executed on any web browsers. Our softwares make use of ESTATE (Energy efficient and Single-state Tweakable block cipher based MAC-Then-Encrypt)algorithm, which is a promising candidate of NIST LWC (National Institute of Standards and Technology LightWeight Cryptography) competition and it provides not only data confidentiality but also data authentication. It also implements the ESTATE algorithm using Web Assembly for efficient use on edge devices, and optimizes the performance of the algorithm using the properties of the underlying block cipher. Several methods are applied to efficiently operate the ESTATE algorithm. We use conditional statements to XOR the extended tweak values during the operation of the ESTATE algorithm. To eliminate this unnecessary process, we use a method of expanding and storing the tweak value through pre-computation. The measured results of the ESTATE algorithm implemented with Web Assembly and the reference C/C++ ESTATE algorithm are compared. ESTATE implemented as Web Assembly is measured in web browsers Chrome, FireFox, and Microsoft Edge. For efficiency on server side, we make use of OpenCL which is parallel computing framework in order to process a number of data simultaneously. In addition, when implementing with OpenCL, using conditional statements causes performance degradation. We eliminated the conditional statement using the loop unrolling method to eliminate the performance degradation. In addition, OpenCL operates by moving the data to be encrypted to the local memory because the local memory has a high operation speed. TweAES-128 and TweAES-128-6, which have the same structure as AES algorithm, can apply the previously existing studied T-table method. In addition, the input value 16-byte is processed in parallel and calculated. In addition, since it may be vulnerable to cache-timing attack, it is safely operated by applying the previously existing studied T-table shuffling method. Our softwares cover the necessary security service from edge devices to servers in edge computing services and they can be easily used for various types of edge computing devices because they are all web-based applications.

Keywords: web; Web Assembly; OpenCL; LWC; fast implementation

check for updates

Citation: Park, B.; Chung, S.S. Efficient Implementation of NIST LWC ESTATE Algorithm Using OpenCL and Web Assembly for Secure Communication in Edge Computing Environment. *Sensors* **2021**, *21*, 1987. https://doi.org/doi:10.3390/s21061987

Academic Editor: Taehong Kim

Received: 31 January 2021
Accepted: 6 March 2021
Published: 11 March 2021

1. Introduction

Existing cloud computing methods provide overall services, such as data processing and transmission in servers and data centers. However, with the increase of users using cloud computing services, the amount of data that has to be processed by the server has increased. So, there is a system load in the process of data processing and transmission. To solve this problem, an edge computing method was created. Edge computing method is a method that processes data in devices, such as smartphones, unlike the method in which servers and data centers handle all services. Edge computing method reduces the system load because it only processes data generated by the device. It is also relatively efficient compared to cloud computing because it collects and processes data on its own. In the case of cloud computing, if a server is paralyzed, it is a fatal blow, but, because edge computing performs its own computing, it can effectively respond to failures. Therefore, we propose a web-based application edge computing method using Web Assembly. The existing edge computing method provides services by processing data sent from a server using a method optimized for hardware, such as ARM, RISC-V, and AVR. However, when edge computing is used using a variety of hardware, there is a disadvantage of having to implement a service and cryptographic algorithm according to each hardware. However, this method can be used generally in PCs (Personal Computer), smartphones, and IoT (Internet of Things) devices that can use web-based applications, such as web browsers and web apps. In addition, there is an advantage that can be used in common in various web-based applications without additional modification on implementation.

In addition, in edge computing method, communication between server and edge computing, communication between edge computing and users, and communication between edge computing will be achieved. For secure data communication, it is necessary to encrypt data and verify that the transmitted data is transmitted without change. So, encryption algorithm and authentication algorithm must be used separately. However, we use the LWC ESTATE (LightWeight Cryptography Energy efficient and Single-stateTweakable block cipher based MAC-Then-Encrypt) algorithm, which can do this process at once. In addition, it provides edge computing service by implementing encryption and authentication service of ESTATE algorithm with Web Assembly, which has better performance than JavaScript for communication using web-based applications.

We propose an efficient implementation of the ESTATE algorithm that uses OpenCL parallel processing to efficiently transfer data through the ESTATE algorithm as a web-based application that provides edge computing services on the server. Even if the server system load is reduced due to the edge computing method, the final processed data is stored on the server. It is a web-based application that provides edge computing services on the server and needs to transfer data using the ESTATE algorithm. Therefore, there is a need for a way to efficiently operate the ESTATE algorithm on the server. Therefore, we applied several additional methods to ensure that the ESTATE algorithm works efficiently for each environment.

Contribution

The contribution of this paper is as follows:

1. Web-based application edge computing method using Web Assembly
   As the number of users using cloud computing services increases, so does the amount of data that must be processed. So, there is a system load in the process of providing the service. So, the edge computing approach was created. The edge computing method transmits and processes data to hardware, such as ARM, RISC-V, and AVR, to reduce system load. However, this method has the disadvantage of having to implement the service differently using each hardware environment and programming language. So, we propose a web-based application edge computing method using Web Assembly. Web Assembly was created to show similar performance to a low-level language. The web-based application edge computing method has the advantage that it can be used in common without additional modification in PCs, smartphones, and IoT devices that can use web-based applications. In addition, the edge computing method communicates data between server and web-base application, web-base application and user, and web-base application. So, the ESTATE

algorithm that can generate the encryption process and tag for authentication at once is implemented using Web Assembly to provide edge computing services. Check how far Web Assembly has caught up with the low-level language in terms of performance. Web Assembly was run on Chrome, FireFox, and Microsoft Edge. At Chrome, FireFox, and Microsoft Edge, Web Assembly is approximately 11%, 10%, 9% slower than Reference C code, TweAES-128-6 is about 5%, 2%, 6% slower, and TweGIFT-128 is about 22%, 54%, and 17% slower than Reference C code.

2. LWC ESTATE parallel processing using OpenCL
   ESTATE (Energy efficient and Single-state Tweakable block cipher based MAC-Then-Encrypt) algorithm is designed to be used in a limited environment, but the data are finally stored on the server. Therefore, ESTATE algorithm optimization is also required in the server. ESTATE algorithm divides AD (Associated Data) and messages into 128-bit blocks, encrypts them one block at a time, and affects the next process, so it cannot process a large amount of data through parallel processing at once. Servers have to send data to multiple platforms, so if they are processed sequentially, the communication speed becomes slow. So, we propose a method of simultaneously generating multiple ciphertexts and tags to be sent to multiple web-based applications for edge computing using OpenCL parallel processing. As a result, the implemented TweAES-128, TweAES-128-6, and TweGIFT-128 implemented in OpenCL showed performance improvement of 6.69×, 7.31×, and 1.47×, respectively, compared to the reference C code.

3. Optimization method for safe and efficient operation of ESTATE algorithm
   ESTATE algorithm uses TweAES-128, TweAES-128-6, and TweGIFT-128. We propose several methods for safe and efficient operation, and apply the previously existing studied methods. In the operation process of TweAES-128, TweAES-128-6, and TweGIFT-128, there is a process of XOR operation by expanding the 4-bit tweak value. TweAES-128 and TweAES-128-6 expand to 8-bit, and TweGIFT-128 expand to 32-bit. However, only 0∼7, 15 are used as 4-bit tweak values. Therefore, we propose a way to store and use 8-bit, 32-bit extended tweak values for 94-bit tweak values through pre-computation. In the OpenCL implementations of TweAES-128, TweAES-128-6, and TweGIFT-128, to remove the performance load, we use a loop unwind method to remove the load and implement it using local memory with a relatively fast working speed. The operation process of TweAES-128 and TweAES-128-6 is the same as AES algorithm. Therefore, the T-table method, which was previously existing studied, was applied. In addition, AES algorithm is vulnerable to cache-timing attack, and TweAES-128 and TweAES-128-6 with the same structure will be vulnerable. Therefore, TweAES-128 and TweAES-128-6 are safely operated by applying the T-table shuffling method, which is the method previously existing studied. TweAES-128 and TweAES-128-6, which applied the table shuffling method previously existing studied, show about 7% and 51% performance overhead, respectively. It simply shuffles the T-table, so it shows little performance overhead.

The remainder of this paper is organized as follows. Section 2 provides a basic overview of the web environment, Web Assembly, OpenCL, Edge computing, and LWC ESTATE. Section 3 describes the relate work of OpenCL and Web Assembly. Section 4 describes the method proposed in the paper. Section 5 describes the performance measurement results. Finally, Section 6 concludes the paper.

2. Background

2.1. Edge Computing

Several companies have used cloud computing methods [1] to provide computing services, such as servers, storage, software, and analytics, over the Internet. Cloud computing is a method focused on centralizing services to several large data centers. However, such cloud computing also begins to have problems. As the number of people using cloud services increases, the amount of data processed by servers and data centers increases, and data delays occur in the process of analyzing and transmitting collected data. That is why edge computing [2–4] was created to solve problems, such as data processing speed, capacity, and security. Edge computing is performing computing at or near the physical location of a user or data source. In the case of cloud computing, data is processed in the data center, whereas edge computing is a method of processing data in devices, such as smartphones. Edge computing method has several advantages. When using cloud computing, the larger the amount of data to be processed, the higher the system load, but in the case of edge computing, data load can be reduced because only data generated by the device is processed. In addition, cloud computing has to strengthen security from the process of data transmission and delivery with a central server architecture, but edge computing is relatively more secure than cloud computing because it collects and processes data on its own. In addition, when the server is paralyzed when using cloud computing, the overall damage is seriously affected, but when using edge computing, it can effectively respond to failures because it performs its own computing. Figure 1 is the structure of the edge computing method. It shows a structure that does not process data in a server or data center but sends data to peripheral devices that will perform edge computing services and sends data to the user's device after processing.

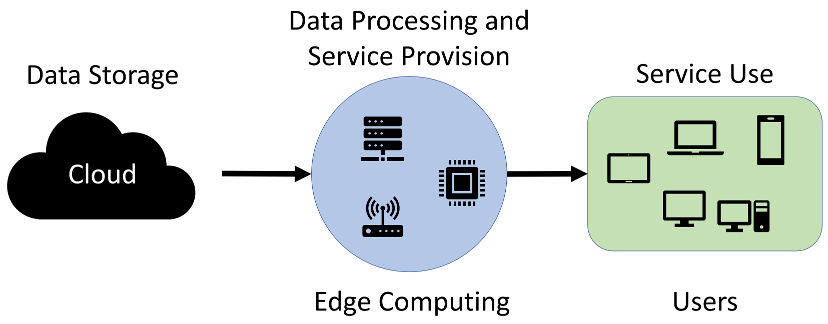

Figure 1. Edge computing structure.

2.2. *Web Environment*

Due to the continuous development of the web environment, various functions are being performed in the web environment. Due to the advancement of web technology, information on the web is displayed the same on different platforms to which networks are connected, such as PCs or smart devices. Web-based applications run within a web browser without communicating with the operating system. Due to the development of internet technology, and hardware performance improve, web technologies and libraries are continuously being created so that more complex and heavy calculations and functions can be made in a web environment. There are various web browsers in which these functions can be used, and various web browsers, such as Chrome, FireFox, and Microsoft Edge, exist. Each web browser has a JavaScript engine that renders JavaScript code and a rendering engine that provides visual services to users through web screens. Chrome uses V8 and Blink, FireFox uses SpiderMonkey and Gecko, and Microsoft Edge uses Chakra and EdgeHTML as JavaScript engines and rendering engines. There is Node.js [5], a software platform used for network application development. Node.js includes a built-in http server library, so it can be operated without additional software on the web server, and through this, more control over the operation of the web server is possible. In addition, web socket communication is possible using Node.js. Figure 2 shows the process of communication between the user who uses the web and the web server, and the process of storing data generated while using the web in the database.

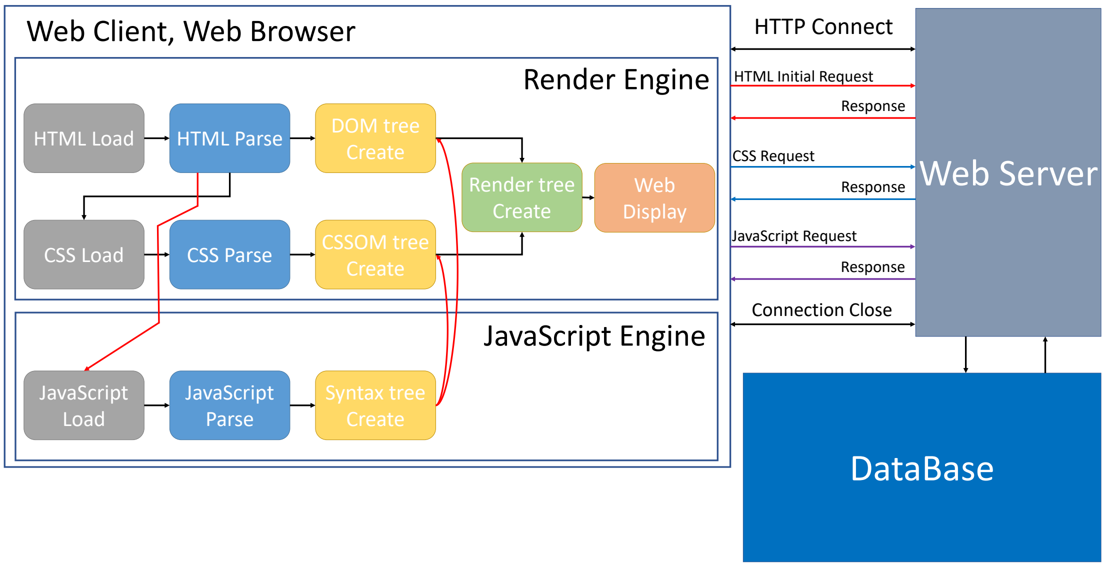

Figure 2. Web operation process [6].

2.3. Web Assembly

With the development of web-based applications, various organizations, companies, and individuals develop web-based applications to provide various services, and various web technologies are being developed. In addition, many users access web-based applications to use the various services and functions provided. These web-based applications are mainly developed in JavaScript, which is a cross-platform language, to display the same information to users on multiple platforms. However, as the number of web users increases, the amount of data that needs to be collected and processed increases, so it is important to increase the speed of processing data to reduce the load. In order to compute faster even in the web environment, Web Assembly [7,8] was created and it is constantly evolving. In addition, Web Assembly can be used by converting to languages with data types, such as C [9], C++ [10], and Typescript [11]. Therefore, it is possible to use previously implemented codes without additional modification. Due to the existence of data types, unlike JavaScript, mathematical operations allow the desired value to be computed without additional computation. Figure 3 shows the process of converting to Web Assembly using programming languages that have data types, such as C, C++, and Rust [12].

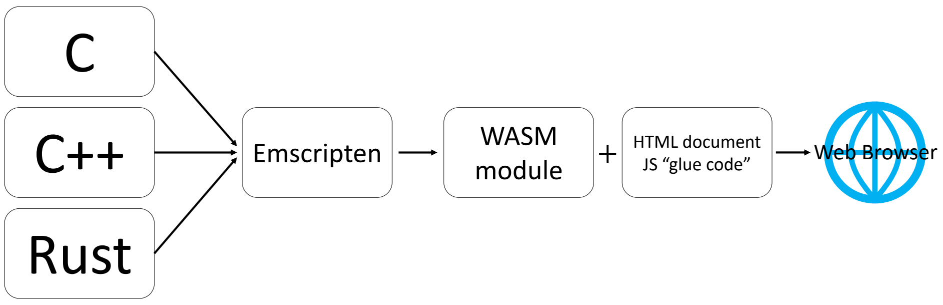

Figure 3. Web Assembly conversion process [6].

2.4. OpenCL

OpenCL is an open general purpose parallel computing framework for creating programs that run on heterogeneous platforms, such as CPUs and GPUs. OpenCL provides task-based, data-based parallel computing. OpenCL can be used in AMD, Intel CPU, Intel integrated graphics, and NVIDIA graphics card products. Table 1 and Figure 4 show the four types of memory used in OpenCL and their respective functions.

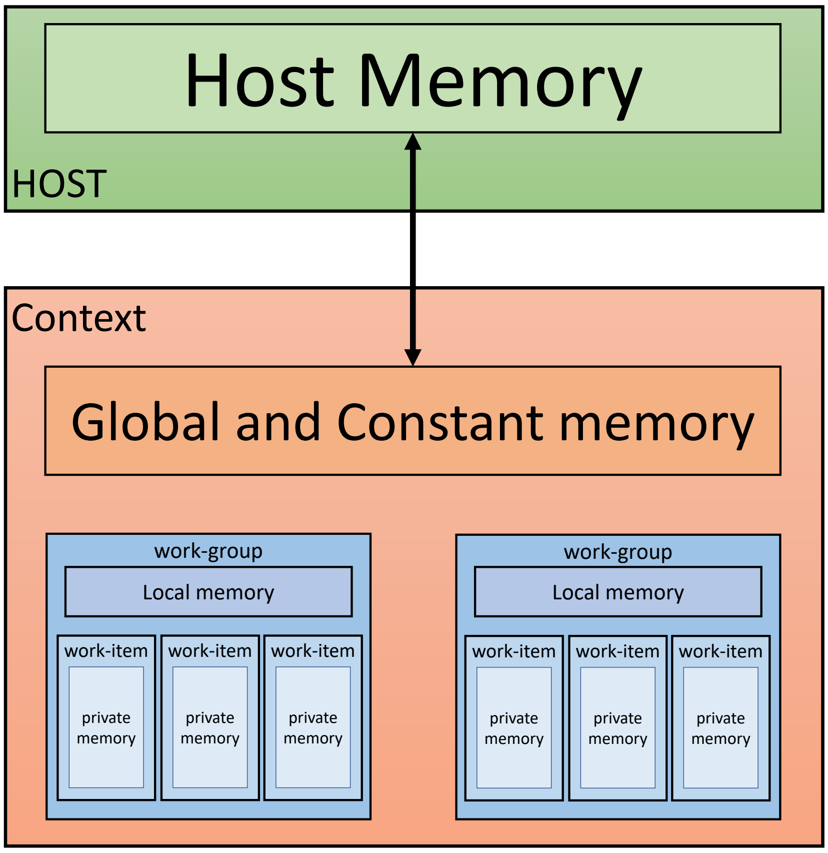

Figure 4. OpenCL memory structure [13].

Figure 5 shows the OpenCL platform model consisting of one host and one or more devices. The OpenCL platform always contains only one host. Each device has one or more compute units, and each computational unit has one or more processing element (PE). The device is where the kernel runs. Devices are provided by CPU, GPU, DSP, and hardware manufacturers. And the actual calculation for the device is done in PE.

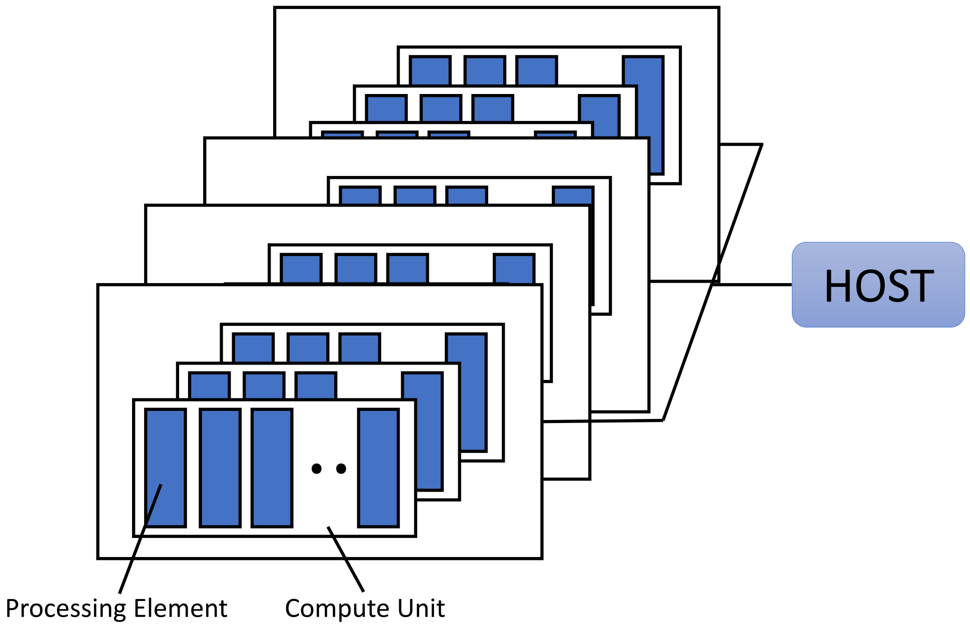

Figure 5. OpenCL platform model [13].

Table 1. OpenCL memory characteristics [14,15].

Memory	Characteristics
Global Memory	(1) Read and write from all work items (2) Placed in device's main memory
Constant Memory	(1) Read only from all work items (2) Placed in device's main memory
Local Memory	(1) Can be shared and used among work items in a work group (2) In many cases, a shared memory disposed on each operation unit is used.
Private Memory	(1) Dedicated memory area for work items (2) Often times you use registers used by processing elements.

2.5. Lightweight Cryptography (Lwc) Estate

Table 2 is a table of notations, operations, and algorithms used in the ESTATE algorithm.

Energy efficient and Single-state Tweakable block cipher-based MAC-Then-Encrypt (ESTATE) [16], one of the second round candidates for lightweight encryption algorithms, adopts FCBC-like [17] authentication and is a tunable block cipher-based authentication encryption system using OFB [18] encryption. ESTATE is based on the MAC-then-Encrypt paradigm [19]. ESTATE does not require field multiplications and has single-state, inverse-free, and RUP secure construction features [16]. In addition, ESTATE is divided into ESTATE mode and sESTATE mode. In ESTATE mode, TweAES-128 and TweGIFT-128 are proposed and used as core algorithms. TweAES-128 and TweGIFT-128 are modified versions of AES-128 [20] and GIFT-128 [21], respectively.

$$\forall X \in \bigcup_{m=1}^{n} 0,1^m, X \mapsto \begin{cases} 0^{n-|X|-1}\|1\|X & if|X| < n, \\ X & otherwise, \end{cases} \tag{1}$$

$$(E_1;E_2)?a:b:c:d := \begin{cases} a & if E_1 \wedge E_2 \\ b & if E_1 \wedge \neg E_2 \\ c & if \neg E_1 \wedge E_2 \\ d & if \neg E_1 \wedge \neg E_2 \end{cases}. \tag{2}$$

If the last block of the message and AD is smaller than 128-bit, padding is performed using Equation (1). Equation (2) is to determine the tweak value used in the ESTATE encryption process.

Table 2. Energy efficient and Single-state Tweakable block cipher-based MAC-Then-Encrypt (ESTATE) notation [16].

Notation	Denote	Notation	Denote
$\|A\|$	length(bit) of A	K	$K \in \{0,1\}^k$
$(X_{k-1},...,X_0) \overset{8}{\leftarrow} x$	n-bit block parsing of X	T	$T \in \{0,1\}^t$
i	$0 \le i \le k-2$	M	$M \in \{0,1\}^m$
$\|X_i\|$	$\|X_i\| = n$	$\tilde{E}$	TweAES-128 or TweGIFT-128
$\|X_{k-1}\|$	$1 \le \|X_{k-1}\| \le n$	$\tilde{F}$	TweAES-128-6
$A \oplus B$	the bitwise XOR of A and B	$X \lll i$	i-bit left
$A\|\|B$	the concatenation of A and B	$X \ggg i$	i-bit right
n	128-bit block size	k	128-bit key size
t	128-bit tag size	τ	4-bit tweak size

Algorithm 1 is the functions used in the overall encryption and decryption operation in ESTATE mode. MAC function [16] is a function that creates a tag. The FCBC* [16] function is a function that determines the tweak value according to AD, message length, and encryption process conditions. OFB function [16] is a function that creates a cipher text using the created Tag value. Figures 6–8 show ESTATE mode operation process when both

AD and Message are used as input values, when only Message is used, and when only AD is used.

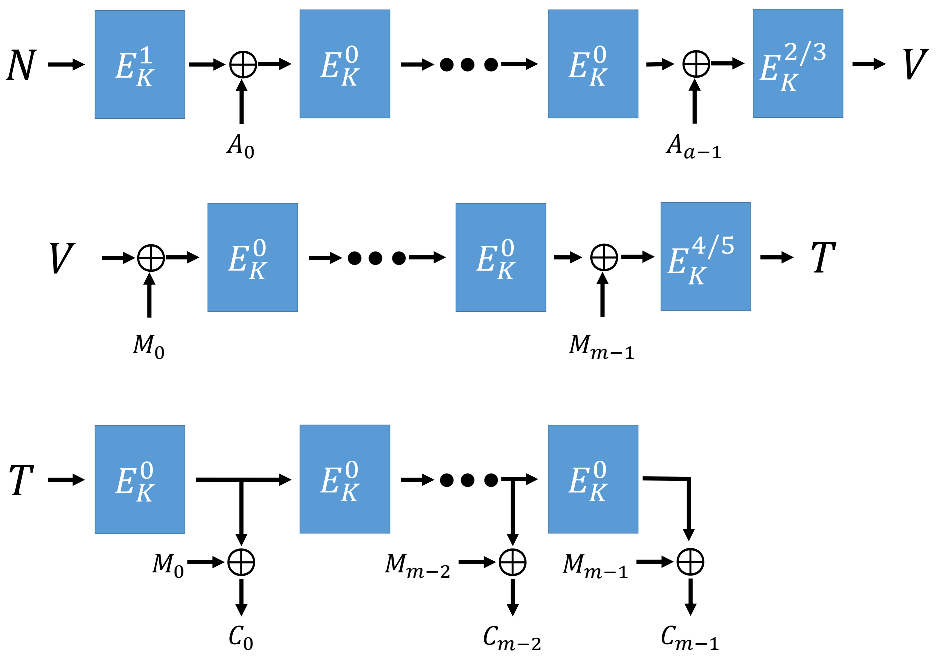

Figure 6. ESTATE mode (Using AD and Message) [16].

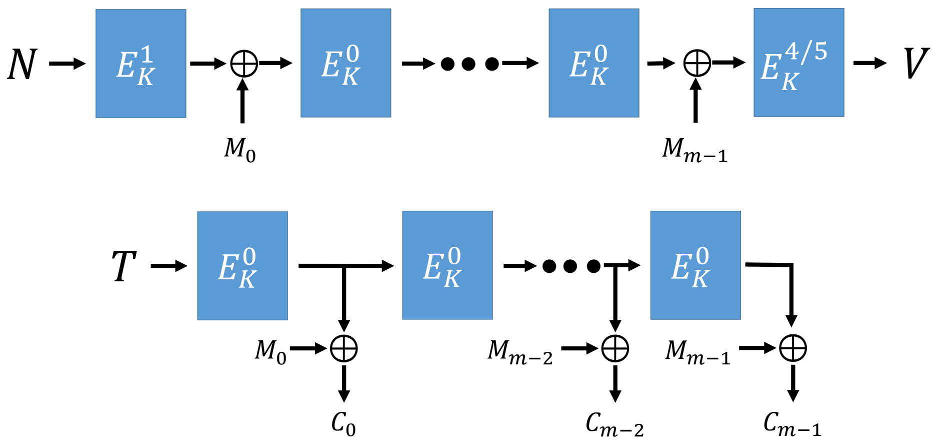

Figure 7. ESTATE mode (no AD, using Message) [16].

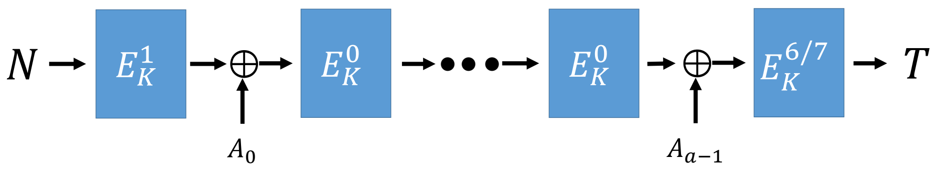

Figure 8. ESTATE mode (using AD, no Message) [16].

Algorithm 2 is the overall operation process of sESTATE mode. sESTATE mode has the same operation process as ESTATE mode. Figures 9–11 show the process of sESTATE mode operation.

Algorithm 1 ESTATE Encryption, Tag Creation, Authentication, and Decryption Algorithm [16].

```
1:  function ESTATE.ENC[$\tilde{E}$]($K, N, A, M$)
2:    $T \leftarrow MAC[\tilde{E}](K, N, A, M)$
3:    $C \leftarrow OFB[\tilde{E}](K, T, M)$
4:    retrun $(C, T)$

5:  function MAC[$\tilde{E}$]($K, N, A, M$)
6:    if $|A| = 0$ and $|M| = 0$ then
7:      return $T \leftarrow \tilde{E}^8_K(N)$
8:    $T \leftarrow \tilde{E}^1_K(N)$
9:    if $|A| > 0$ then
10:     $A_{a-1} \| \cdots \| A_0 \leftarrow A$
11:     $t \leftarrow (|M| > 0\,;|A_{a-1}| = n)\ ?\ 2:3:6:7$
12:     $T \leftarrow$ FCBC$^*[\tilde{E}](K, T, M, t)$
13:   if $|M| > 0$ then
14:     $M_{m-1} \| \cdots \| M_0 \leftarrow M$
15:     $t \leftarrow (|M_{m-1}| = n)\ ?\ 4:5$
16:     $T \leftarrow$ FCBC$^*[\tilde{E}](K, T, M, t)$
17:   return $T$

18: function ESTATE.DEC[$\tilde{E}$]($K, N, A, C, T$)
19:   $M \leftarrow OFB[\tilde{E}](K, T, C)$
20:   $T' \leftarrow MAC[\tilde{E}](K, N, A, M)$
21:   return $(T' = T)\ ?\ M : \perp$

22: function FCBC$^*[\tilde{E}](K, T, D, t)$
23:   $D_{d-1} \| \cdots \| D_0 \leftarrow D$
24:   for $i = 0$ to $d - 2$ do
25:     $T \leftarrow \tilde{E}^0_K(T \oplus D_i)$
26:   $T \leftarrow \tilde{E}^t_K(T \oplus$ ozp$(Dd - 1))$
27:   return $T$

28: function OFB[$\tilde{E}$]($K, T, M$)
29:   $M_{m-1} \| \cdots \| M_0 \leftarrow M$
30:   for $i = 0$ to $m - 1$ do
31:     $T \leftarrow \tilde{E}^0_K(T)$
32:     $C_i \leftarrow$ chop$(T, |M_i|) \oplus M_i$
33:   return $(C_{m-1} \| \cdots \| C_0)$
```

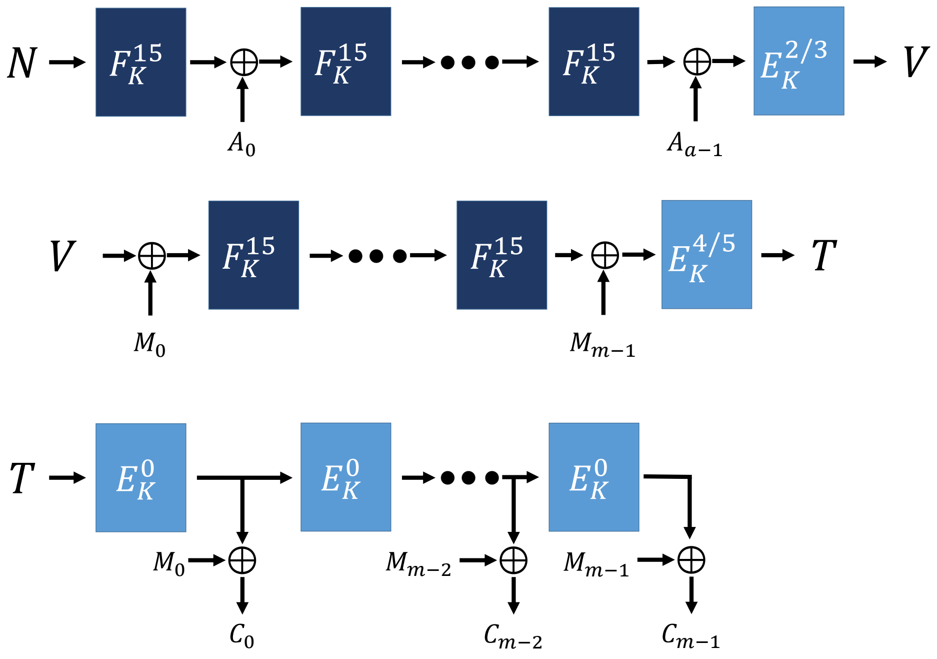

Figure 9. sESTATE mode (using AD and Message) [16].

Algorithm 2 sESTATE Encryption, Tag Creation, Authentication, and Decryption Algorithm [16].

```
1:  function ESTATE.ENC$[\tilde{E},\tilde{F}](K,N,A,M)$
2:      $T \leftarrow MAC[\tilde{E},\tilde{F}](K,N,A,M)$
3:      $C \leftarrow OFB[\tilde{E}](K,T,M)$
4:      retrun $(C,T)$

5:  function MAC$[\tilde{E},\tilde{F}](K,N,A,M)$
6:      if $|A| = 0$ and $|M| = 0$ then
7:          return $T \leftarrow \tilde{E}^{8}_{K}(N)$
8:      $T \leftarrow \tilde{F}^{15}_{K}(N)$
9:      if $|A| > 0$ then
10:         $A_{a-1}\|\cdots\|A_0 \leftarrow A$
11:         $t \leftarrow (|M| > 0\,;\,|A_{a-1}| = n)\ ?\ 2:3:6:7$
12:         $T \leftarrow$ FCBC$^*[\tilde{E},\tilde{F}](K,T,M,t)$
13:     if $|M| > 0$ then
14:         $M_{m-1}\|\cdots\|M_0 \leftarrow M$
15:         $t \leftarrow (|M_{m-1}| = n)\ ?\ 4:5$
16:         $T \leftarrow$ FCBC$^*[\tilde{E},\tilde{F}](K,T,M,t)$
17:     return $T$

18: function ESTATE.DEC$[\tilde{E},\tilde{F}](K,N,A,C,T)$
19:     $M \leftarrow OFB[\tilde{E}](K,T,C)$
20:     $T' \leftarrow MAC[\tilde{E},\tilde{F}](K,N,A,M)$
21:     return $(T' = T)\ ?\ M : \perp$

22: function FCBC$^*[\tilde{E},\tilde{F}](K,T,D,t)$
23:     $D_{d-1}\|\cdots\|D_0 \leftarrow D$
24:     for $i = 0$ to $d-2$ do
25:         $T \leftarrow \tilde{F}^{15}_{K}(T \oplus D_i)$
26:     $T \leftarrow \tilde{E}^{t}_{K}(T \oplus \mathrm{ozp}(Dd-1))$
27:     return $T$

28: function OFB$[\tilde{E}](K,T,M)$
29:     $M_{m-1}\|\cdots\|M_0 \leftarrow M$
30:     for $i = 0$ to $m-1$ do
31:         $T \leftarrow \tilde{E}^{0}_{K}(T)$
32:         $C_i \leftarrow \mathrm{chop}(T,|M_i|) \oplus M_i$
33:     return $(C_{m-1}\|\cdots\|C_0)$
```

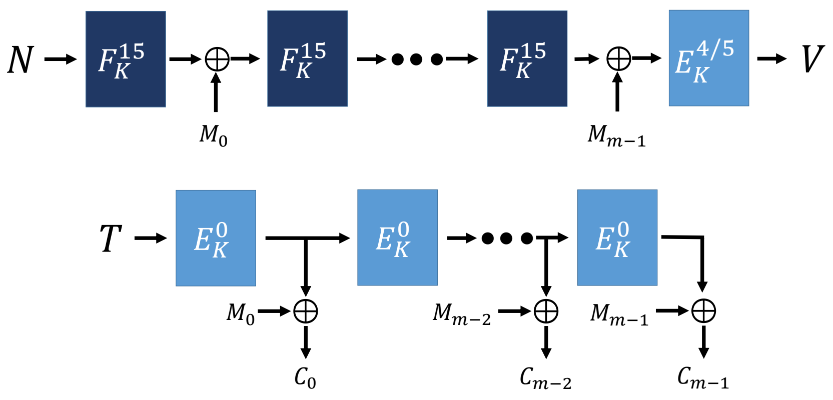

Figure 10. sESTATE mode (no AD, using Message) [16].

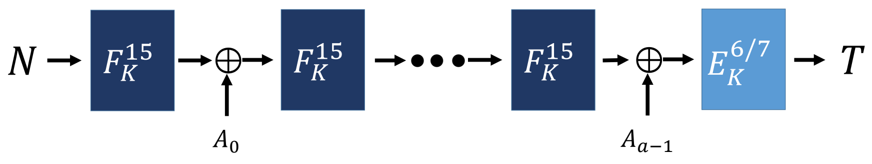

Figure 11. sESTATE mode (using AD, no Message) [16].

2.5.1. TweAES-128, TweAES-128-6

Algorithm 3 are functions of TweAES-128, a cryptographic algorithm used in ESTATE mode. The overall process is the same as AES-128, and a process of XOR operation is added by expanding the 4-bit tweak value to an 8-bit tweak value for every even round except the last round. TweAES-128-6 is proposed and used as a cryptographic algorithm to be used while designing the sESTATE mode in the ESTATE algorithm. TweAES-128-6 has the same operation process as TweAES-128. The difference is that the TweAES-128 runs 10 rounds, while the TweAES-128-6 only runs 6 rounds.

Algorithm 3 TweAES-128 Algorithm [16].

```
function TweAES($K, T, M$)
    $(W_{43}, ..., W_0) \leftarrow$ KeyGen($K$)
    $X \leftarrow X \oplus (W_3, W_2, W_1, W_0)$
    for $i = 1$ to 9 do
        $X \leftarrow$ SubBytes($X$)
        $X \leftarrow$ ShiftRows($X$)
        $X \leftarrow$ MixColumns($X$)
        $X \leftarrow X \oplus (W_{4i+3}, W_{4i+2}, W_{4i+1}, W_{4i})$
        if $i\%2 = 0$ then
            $X \leftarrow$ AddTweak($X, T$)
    $X \leftarrow$ SubBytes($X$)
    $X \leftarrow$ ShiftRows($X$)
    $X \leftarrow X \oplus (W_{43}, W_{42}, W_{41}, W_{40})$
    return $X$

function TweAES-6($K, T, X$)
    $(W_{43}, ..., W_0) \leftarrow$ KeyGen($K, X$)
    $X \leftarrow X \oplus (W_3, W_2, W_1, W_0)$
    for $i = 1$ to 6 do
        $X \leftarrow$ SubBytes($X$)
        $X \leftarrow$ ShiftRows($X$)
        $X \leftarrow$ MixColumns($X$)
        $X \leftarrow X \oplus (W_{4i+3}, W_{4i+2}, W_{4i+1}, W_{4i})$
        if $i\%2 = 0$ and $i < 6$ then
            $X \leftarrow$ AddTweak($X, T$)
    return $X$

function AddTweak($X, T$)
    $(X_{127}, ..., X_0) \xleftarrow{1} X$
    $(T_3, ..., T_0) \xleftarrow{1} T$
    $T_\oplus \leftarrow T_0 \oplus T_1 \oplus T_2 \oplus T_3$
    for $i = 0$ to 3 do
        $T_{i+4} \leftarrow T_i \oplus T_\oplus$
    for $i = 0$ to 7 do
        $X_{8i} \leftarrow X_{8i} \oplus T_i$
    return $X$
```

2.5.2. TweGIFT-128

Algorithm 4 is the overall process of TweGIFT-128 used in ESTATE mode. TweGIFT-128 has the same structure as GIFT-128, and XOR operation is added by expanding the 4-bit tweak value to 32-bit tweak value every $(round + 1)\%5 == 0$th rounds. TweGIFT-128's tweak expansion process is the same as TweAES-128. First, expand it to 8-bit in the same way, and then use the expanded 8-bit value to store the same value in the remaining 24-bits and expand it to a total of 32-bits.

Algorithm 4 TweGIFT-128 Algorithm [16].

```
function TweGIFT($K, T, X$)
    $C \leftarrow 000000$
    for $i = 0$ to 39 do
        $X \leftarrow$ SubCells($X$)
        $X \leftarrow$ PermBits($X$)
        $(K, X) \leftarrow$ AddRoundKey($K, X$)
        $(C, X) \leftarrow$ AddRoundConstant($C, X$)
        if $(i + 1)\%5 = 0$ and $i < 39$ then
            $X \leftarrow$ AddTweak($X, T$)
    return $X$

function AddTweak($X, T$)
    $(X_{127}, ..., X_0) \xleftarrow{1} X$
    $(T_3, ..., T_0) \xleftarrow{1} T$
    $T_\oplus \leftarrow T_0 \oplus T_1 \oplus T_2 \oplus T_3$
    for $i = 0$ to 3 do
        $T_{i+4} \leftarrow T_i \oplus T_\oplus$
    $T_{15..8} \leftarrow T_{7..0}$
    $T_{23..16} \leftarrow T_{7..0}$
    $T_{31..24} \leftarrow T_{7..0}$
    for $i = 0$ to 31 do
        $X_{4i} \leftarrow X_{4i} \oplus T_i$
    return $X$
```

3. Related Work

3.1. Existing Crypto Implementation Using OpenCL

Due to the development of multi-core processes, parallel processing technology is being used in various fields. In addition, the use of OpenCL for parallel processing is increasing, and it is efficient for processing large amounts of data. Therefore, studies are being conducted to rapidly encrypt a large amount of data using an cryptographic algorithm using OpenCL.

In Reference [22], we use OpenCL to improve encryption speed using the AES encryption algorithm. They used the NVIDIA GeForce GTX 1060 to measure performance. Table 3 is a table comparing the results measured in Reference [22] with previous studies. As a result, their research results show that the XTS (XEX-based tweaked-codebook mode with ciphertext stealing) mode is 12.86% and the CTR (Counter) mode is 14.71%, compared to the previous studies.

Table 3. AES (Advanced Encryption Standard) fast implementation study results comparison.

Paper	GPU	Language	Mode	Throughput (Gbps)
Yuan et al. [23]	ATI HD 7670M	OpenCL	CTR	5.04 Gbps
Wang et al. [24]	NVIDIA GTX 285	OpenCL	XTS	8.59 Gbps
Wang et al. [24]	NVIDIA GTX 285	CUDA	XTS	9.74 Gbps
Conti et al. [25]	NVIDIA GT 555M	OpenCL	CTR	10.00 Gbps
Biagio et al. [26]	NVIDIA GT 8800	CUDA	CTR	12.50 Gbps
Sanida et al. [22]	NVIDIA GTX 1060	OpenCL	XTS	12.53 Gbps
Sanida et al. [22]	NVIDIA GTX 1060	OpenCL	CTR	14.71 Gbps

In Reference [27], various cryptographic algorithms are implemented in OpenCL and used for image encryption. Table 4 is an information table that implements AES (Advanced Encryption Standard), DES (Data Encryption Standard), BlowFish, and RSA (Ron Rivest, Adi Shamir, Leonard Adleman) using OpenCL in Reference [27]. Table 5 is the result of measurement by CPU and GPU for each cryptographic algorithm implemented using OpenCL. As a result, AES, DES, BlowFish, and RSA show performance improvements of 8 times, 2.5 times, 11.13 times, and 5 times, respectively.

Table 4. Memory size, line of code for cryptographic algorithm implementation using OpenCL [27].

Cryptographic Algorithm	Key Size	Constant Space	Compilation Time
AES [20]	128-bit	844 KB	2.7 ms
DES [28]	192-bit	1294 KB	5.3 ms
BlowFish [29]	256-bit	252 B	3.5 ms
RSA [30]	128-bit	6 KB	1031 ms

Table 5. Measuring cryptographic algorithm results using OpenCL [27].

Device	AES	DES	BlowFish	RSA
AMD FX 6100 3.0 GHz (CPU 6 Cores)	240 Mbps	144 Mbps	736 Mbps	4 Mbps
NVIDIA GTX 550 (GPU)	1920 Mbps	368 Mbps	8192 Mbps	20 Mbps
Ratio of Performance Improvement	8 times	2.5 times	11.13 times	5 times

In Reference [31], AES-256 encryption and decryption implementation using OpenCL parallel processing is compared with AES-256 implemented using sequential processing. As a result, when 10,240,000 work items are used, the implementation using OpenCL parallel processing shows performance improvement of about 240 times for encryption and 481 times for decryption. In addition, as measured by AMD Radeon HD 8850M and AMD

Radeon HD 8570, AMD Radeon HD 8570 shows performance improvement of 3.8 times and 3.3 times in encryption and decryption, respectively.

3.2. Web Assembly

Web Assembly shows better performance than JavaScript in web-based applications, and due to continuous development, it will continue to be close to the performance of low-level languages, such as C language. In addition, research on Web Assembly is actively underway.

In Reference [6], the revised CHAM, *P*-256-wNAF (window Non-Adjacent Form), SHA-256 (Secure Hash Algorithm), and HMAC (Hash-based Message Authentication Code) algorithms are compared after implementation using Web Assembly and JavaScript for more efficient encryption, key exchange, and authentication in the web environment. Table 6 shows the performance measurement results for cryptographic algorithms, and it can be seen that it is more efficient when Web Assembly implements cryptographic algorithms than JavaScript. In addition, in the case of wNAF used for key exchange, the Atomic block method was applied to be safe from side-channel attack (SCA) [32]. Web Assembly shows that it can operate efficiently and safely because its performance overhead ratio is lower than that of JavaScript.

Table 6. Web Assembly and JavaScript performance measurement and comparison through cryptographic algorithm implementation (cpb: Cycle Per Byte) [6].

	Chrome		**FireFox**		**Microsoft Edge**	
	Web Assembly	**JavaScript**	**Web Assembly**	**JavaScript**	**Web Assembly**	**JavaScript**
revised CHAM-64/128 [33]	120 cpb (2.1 times)	260 cpb	120 cpb (2.1 times)	260 cpb	120 cpb (2 times)	240 cpb
revised CHAM-128/128 [33]	60 cpb (3 times)	180 cpb	60 cpb (1.6 times)	100 cpb	70 cpb (1.8 times)	130 cpb
revised CHAM-128/256 [33]	70 cpb (3 times)	210 cpb	70 cpb (2.1 times)	150 cpb	70 cpb (2.8 times)	200 cpb
wNAF	27 cpb (11 times)	300 cpb	30 cpb (12 times)	365 cpb	27 cpb (11 times)	322 cpb
wNAF [34] (Atomic block [35])	42 cpb (10 times)	447 cpb	37 cpb (10 times)	405 cpb	37 cpb (14 times)	522 cpb
wNAF (Improved Atomic block [6])	32 cpb (11 times)	365 cpb	32 cpb (12 times)	387 cpb	30 cpb (14 times)	437 cpb
SHA-256 [36]	27 cpb (7.5 times)	203 cpb	20 cpb (10.8 times)	216 cpb	20 cpb (11 times)	221 cpb
HMAC [37]	92 cpb (7.5 times)	697 cpb	93 cpb (24.8 times)	2315 cpb	97 cpb (7.1 times)	693 cpb

Reference [38] converts the Picnic algorithm [39] to Web Assembly, measures the performance in Chrome, FireFox, and Microsoft Edge, and compares it with the C/C++ implementation. As a result, the Picnic algorithm implemented by Web Assembly is about 2∼3 times slower than the C/C++ implementation.

3.3. Cache Timing Attack

There are various attack methods, such as differential attack and side-channel attack, to find out important information about encryption algorithm. In addition, there is an attack method that finds out the key value, which is important information of the cryptographic algorithm through the cache access time of the CPU, and research on this is being actively studied as interest in it increases. Ref. [40] proved the vulnerability through an attack to find the last round key against the T-table AES algorithm of OpenSSL 1.1.0f [41]. So, in

Reference [40], they study and apply the T-table shuffling method to be safe against Flush + Reload, a kind of cache-timing attack [42]. In Reference [40], they randomly shuffle the array containing values from 0 to 255 using the Fisher-Yates function [43]. Then, the values stored in 4 256-byte T-tables are shuffled and used by using the shuffled array values. In Reference [40], the T-table was shuffled using the Fisher-Yates function in the AES T-table, and the test shows that it is safe against Flush + Reload cache-timing attacks.

4. Proposed Implementation for Secure Communication in Edge Computing Services

4.1. Overall Architecture of Proposed Software

The existing edge computing method processes data received from a server or user or data to be sent, and communicates through encryption and authentication. Therefore, in hardware, such as ARM, AVR, and RISC-V used in edge computing for encryption and authentication, secure communication is implemented by implementing encryption algorithms and authentication algorithms using programming languages suitable for each environment. However, since each environment uses different performance, different functions, and different programming languages, even the same algorithm needs to be implemented in each hardware. So, we use Web Assembly to implement encryption and authentication so that it can be used generally on each device. In addition, it uses the LWC ESTATE algorithm, which has both an encryption function and an authentication process. Web Assembly is designed for performance similar to a low-level language in a web environment. The ESTATE algorithm implemented by Web Assembly can be used in general without additional modification in PCs, smartphones, and IoT devices where web apps and web browsers can be used. Therefore, once created, it can be used in multiple devices for secure communication. In addition, the finally processed data is stored on the main server. Therefore, we propose additional optimization methods to use the ESTATE algorithm efficiently in the server. The operation process of the ESTATE algorithm has a characteristic that affects the next process using the previous value. Therefore, it is difficult to process a large amount of data at the same time. However, if the main server processes data sequentially, even if the edge computing method is used, the communication process eventually shows slow performance. So, we propose a method of using OpenCL parallel processing so that multiple ciphertexts and tags to be sent to multiple web-based applications can be created at the same time. In addition, to safely and efficiently operate the ESTATE algorithm, an additional method is proposed, and the previously existing studied methods are applied. During operation of TweAES-128, TweAES-128-6, and TweGIFT-128 used in the ESTATE algorithm, the 4-bit tweak value is checked for each specific round through conditional statements, and then expanded to perform XOR (exclusive OR) operation on the encrypted data. Therefore, we propose a method of storing and using the extended tweak values for 9 4-bit tweak values through pre-computation. So, tweak values are extended to 8-bit and 32-bit, respectively, through pre-computation. In the implementation of OpenCL, if there is a conditional statement, there is a load in the operation process. The ESTATE algorithm uses conditional statements due to the type of input value, tweak value check for each specific round, and tweak value XOR operation for each specific round. So, when we implement TweAES-128, TweAES-128-6, and TweGIFT-128 using OpenCL, we implement it using the loop unrolling method to eliminate performance degradation. In addition, it operates using local memory, which has a high operation speed. TweAES-128 and TweAES-128-6 are similar in operation to the AES algorithm. Therefore, it operates faster by applying the existing T-table method. In addition, there are studies that the AES algorithm is vulnerable to cache-timing attacks. Since TweAES-128 and TweAES-128-6, which have the same structure as the AES algorithm, can be vulnerable, they are safely operated by applying the T-table shuffling method, which is an the existing cache-timing attack response algorithm.

4.2. Edge Computing and Estate Implementation Using Web Assembly

We propose a web-based application edge computing method using Web Assembly. Web Assembly was created to show performance similar to low-level language in web environment. The existing edge computing method provides services by optimizing each environment and functions in hardware, such as ARM, AVR, and RISC-V. However, this method is difficult to use in general because it uses programming languages and functions used in each environment, such as ARM, AVR, and RISC-V, and additional cost is consumed because additional implementation is required for each device. The web-based application edge computing method proposed by us can be used in PCs, smartphones, IoT devices, etc. that can basically use web-based applications. In addition, even if the platform is different, it is efficient because it can be used generally without additional modification in terms of implementation. In addition, in order to implement the algorithm with Web Assembly, the existing code implemented in a programming language with a data type can be converted and used, so there is no additional cost. In addition, if you use a library, such as Node.js, so that web socket communication is possible without adapting the communication process to each hardware, communication becomes easy. Web-based application In the edge computing method, communication between server and web-based application, communication between web-based application and user, and communication between web-based application are made. Encryption and authentication functions are required to safely send data in various communication processes. So, we use the ESTATE algorithm, which has encryption and authentication functions. Therefore, as shown in Figure 12, in a web-based application using Web Assembly, a ciphertext and a tag for authentication are created using the ESTATE algorithm, and data is safely delivered to the user.

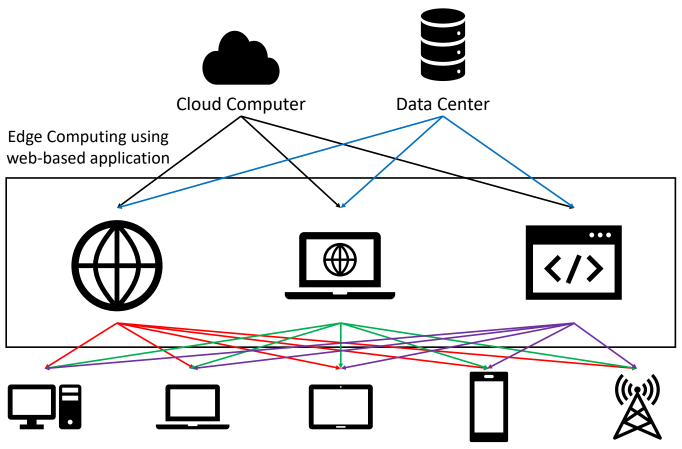

Figure 12. Edge computing structure using Web Assembly.

4.3. Parallel Implementation of Estate Using OpenCL

The ESTATE algorithm uses TweAES-128 and TweGIFT-128 to encrypt each block of 128-bit size. Then, the next step is performed using the previously encrypted result value. Therefore, it is impossible to use a method of processing a large amount of data at once through parallel processing. It is designed for use in a limited environment, but the finally communicated data is stored on the server. Therefore, it is necessary to implement ESTATE according to the server environment so that the server can use ESTATE efficiently. We use OpenCL to simultaneously calculate and transmit ciphertext and tag generation to be sent to multiple web-based applications.

Instead of sequentially processing multiple data using the ESTATE algorithm, it uses a method of simultaneously processing using OpenCL parallel processing as shown in Figure 13. When implemented using OpenCL parallel processing, performance degradation occurs when conditional statements exist. TweAES-128, TweAES-128-6, and TweGIFT-128 use conditional statements to check the type of input value, check whether or not padding, check the tweak value, and perform the extended tweak value XOR operation for each round. We use the loop unrolling method to remove the conditional statement in order to remove the performance load in the OpenCL implementation. In addition, the local memory has the fastest operation speed among OpenCL memories. For this reason, data is moved to local memory and encrypted to improve performance. Algorithm 5 is an OpenCL code algorithm that reduces the performance load by eliminating conditional statements using a loop unrolling method.

Algorithm 5 TweAES-128, TweAES-128-6, TweGIFT-128 proposed by applying loop unrolling method.

```
function loop unrolling TweAES-128($K, T, X$)
  $(W_{43}, ..., W_0) \leftarrow$ KeyGen($K, X$)
  $X \leftarrow X \oplus (W_3, W_2, W_1, W_0)$
  for $i$ = 1 to 4 do
    $X \leftarrow$ SubBytes($X$)
    $X \leftarrow$ ShiftRows($X$)
    $X \leftarrow$ MixColumns($X$)
    $X \leftarrow X \oplus (W_{4i+3}, W_{4i+2}, W_{4i+1}, W_{4i})$

    $X \leftarrow$ SubBytes($X$)
    $X \leftarrow$ ShiftRows($X$)
    $X \leftarrow$ MixColumns($X$)
    $X \leftarrow X \oplus (W_{8i+3}, W_{8i+2}, W_{8i+1}, W_{8i})$
    AddTweak(X, T)

  $X \leftarrow$ SubBytes($X$)
  $X \leftarrow$ ShiftRows($X$)
  $X \leftarrow$ MixColumns($X$)
  $X \leftarrow X \oplus (W_{39}, W_{38}, W_{37}, W_{36})$
  $X \leftarrow$ SubBytes($X$)
  $X \leftarrow$ ShiftRows($X$)
  $X \leftarrow X \oplus (W_{43}, W_{42}, W_{41}, W_{40})$

function loop unrolling TweAES-6($K, T, X$)
  $(W_{43}, ..., W_0) \leftarrow$ KeyGen($K, X$)
  $X \leftarrow X \oplus (W_3, W_2, W_1, W_0)$
  for $i$ = 1 to 2 do
    $X \leftarrow$ SubBytes($X$)
    $X \leftarrow$ ShiftRows($X$)
    $X \leftarrow$ MixColumns($X$)
    $X \leftarrow X \oplus (W_{4i+3}, W_{4i+2}, W_{4i+1}, W_{4i})$

    $X \leftarrow$ SubBytes($X$)
    $X \leftarrow$ ShiftRows($X$)
    $X \leftarrow$ MixColumns($X$)
    $X \leftarrow X \oplus (W_{8i+3}, W_{8i+2}, W_{8i+1}, W_{8i})$
    AddTweak(X, T)

  $X \leftarrow$ SubBytes($X$)
  $X \leftarrow$ ShiftRows($X$)
  $X \leftarrow$ MixColumns($X$)
  $X \leftarrow X \oplus (W_{23}, W_{22}, W_{21}, W_{20})$
  $X \leftarrow$ SubBytes($X$)
  $X \leftarrow$ ShiftRows($X$)
  $X \leftarrow X \oplus (W_{43}, W_{42}, W_{41}, W_{40})$

function loop unrolling TweGIFT-128($K, T, X$)
  $C \leftarrow$ 000000
  for $i$ = 0 to 7 do
    for $j$ = 0 to 3 do
      $X \leftarrow$ SubCells($X$)
      $X \leftarrow$ PermBits($X$)
      $(K, X) \leftarrow$ AddRoundKey($K, X$)
      $(C, X) \leftarrow$ AddRoundConstant($C, X$)

    $X \leftarrow$ SubCells($X$)
    $X \leftarrow$ PermBits($X$)
    $(K, X) \leftarrow$ AddRoundKey($K, X$)
    $(C, X) \leftarrow$ AddRoundConstant($C, X$)
    AddTweak(X, T)

    for $i$ = 35 to 39 do
    $X \leftarrow$ SubCells($X$)
    $X \leftarrow$ PermBits($X$)
    $(K, X) \leftarrow$ AddRoundKey($K, X$)
    $(C, X) \leftarrow$ AddRoundConstant($C, X$)
```

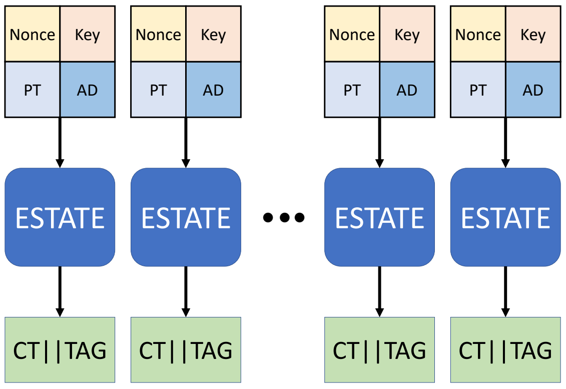

Figure 13. Structure of ESTATE algorithm operation using parallel processing.

4.4. Safe and Efficient Implementation of TweAES-128, TweAES-128-6, TweGIFT-128 of Estate Algorithm

TweAES-128 and TweGIFT-128 are used in ESTATE mode, and TweAES-128-6 is used in sESTATE mode. TweAES-128, TweGIFT-128, and TweAES-128-6 have the same operation process as AES-128 and GIFT-128, but additionally, the process of XOR operation by expanding the 4-bit tweak value is added. However, in TweAES-128, TweAES-128-6, and TweGIFT-128, only 0∼7, 15 are used as tweak values. Therefore, we propose a method to extend the 4-bit tweak value to 8-bit and 32-bit in advance to fit each algorithm and use it after storage. This method eliminates the unnecessary process of repeatedly checking and expanding tweak value. In addition, TweAES-128 and TweAES-128-6 have the same structure as the AES algorithm, so the existing studied T-table method to quickly compute AES can be applied. In addition, it is possible to perform faster operation by processing the 16-byte input value used in both algorithms in parallel.

As shown in Figure 14, the operation process of TweAES-128 and TweAES-128-6 used in the ESTATE algorithm uses an efficient method of simultaneously calculating 16-byte input values through OpenCL parallel processing. In addition, T-table shuffling method, which is the method studied in Reference [40], is applied to the T-table used in ESTATE TweAES-128 and TweAES-128-6 to safely operate against cache-timing attack.

Using method in Reference [40], mix the index value of 0∼255 to shuffle the T-table. Then, the T-table is shuffled using the mixed index value. Algorithm 6 is a process that will be used every round of ESTATE TweAES-128 and TweAES-128-6.

Algorithm 6 ESTATE TweAES-128, TweAES-128-6 Proposal Method Applying T-table Shuffling

```
1: Te0-sf : Te0[shuffle-array]
2: Te1-sf : Te1[shuffle-array]
3: Te2-sf : Te2[shuffle-array]
4: Te3-sf : Te3[shuffle-array]

5: function 1-round(S0∼S3, RK)
6:     S0 = Te0-sf[S0 ≫ 24] ⊕ Te1-sf[S1 ≫ 16 & 0xff] ⊕ Te2-sf[S2 ≫ 8 & 0xff] ⊕ Te3-sf[S3 & 0xff] ⊕ $RK$
7:     S1 = Te0-sf[S1 ≫ 24] ⊕ Te1-sf[S2 ≫ 16 & 0xff] ⊕ Te2-sf[S3 ≫ 8 & 0xff] ⊕ Te3-sf[S0 & 0xff] ⊕ $RK$
8:     S0 = Te0-sf[S2 ≫ 24] ⊕ Te1-sf[S3 ≫ 16 & 0xff] ⊕ Te2-sf[S0 ≫ 8 & 0xff] ⊕ Te3-sf[S1 & 0xff] ⊕ $RK$
9:     S0 = Te0-sf[S3 ≫ 24] ⊕ Te1-sf[S0 ≫ 16 & 0xff] ⊕ Te2-sf[S1 ≫ 8 & 0xff] ⊕ Te3-sf[S2 & 0xff] ⊕ $RK$

10: function 1-round with AddTweak(S0∼S3, RK, tweak)
11:     S0 = Te0-sf[S0 ≫ 24] ⊕ Te1-sf[S1 ≫ 16 & 0xff] ⊕ Te2-sf[S2 ≫ 8 & 0xff] ⊕ Te3-sf[S3 & 0xff] ⊕ $RK$
12:     S1 = Te0-sf[S1 ≫ 24] ⊕ Te1-sf[S2 ≫ 16 & 0xff] ⊕ Te2-sf[S3 ≫ 8 & 0xff] ⊕ Te3-sf[S0 & 0xff] ⊕ $RK$
13:     S0 = Te0-sf[S2 ≫ 24] ⊕ Te1-sf[S3 ≫ 16 & 0xff] ⊕ Te2-sf[S0 ≫ 8 & 0xff] ⊕ Te3-sf[S1 & 0xff] ⊕ $RK$
14:     S0 = Te0-sf[S3 ≫ 24] ⊕ Te1-sf[S0 ≫ 16 & 0xff] ⊕ Te2-sf[S1 ≫ 8 & 0xff] ⊕ Te3-sf[S2 & 0xff] ⊕ $RK$
15:     AddTweak(S0∼S3, tweak)
```

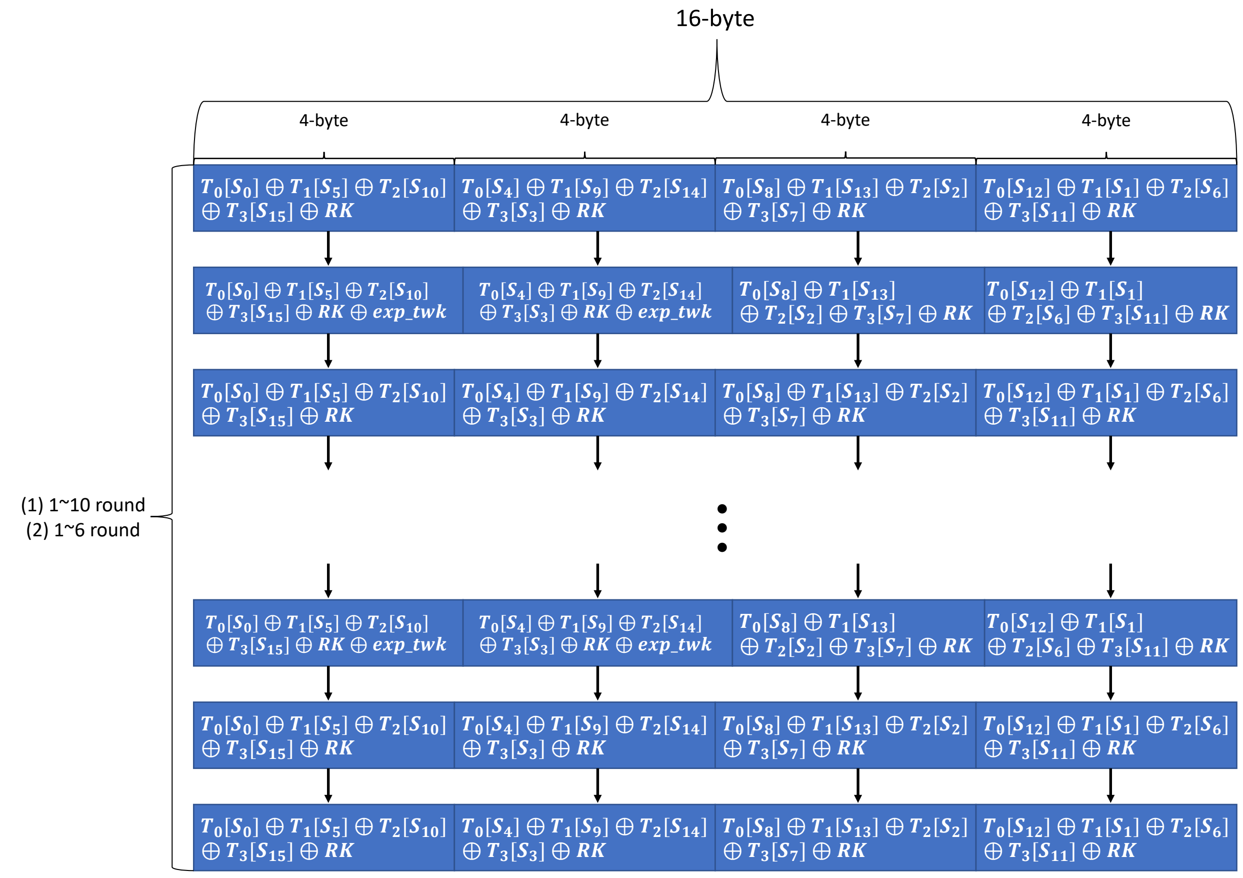
16-byte
4-byte
4-byte
4-byte
4-byte
(1) 1~10 round
(2) 1~6 round
$T_0[S_0] \oplus T_1[S_5] \oplus T_2[S_{10}] \oplus T_3[S_{15}] \oplus RK$
$T_0[S_4] \oplus T_1[S_9] \oplus T_2[S_{14}] \oplus T_3[S_3] \oplus RK$
$T_0[S_8] \oplus T_1[S_{13}] \oplus T_2[S_2] \oplus T_3[S_7] \oplus RK$
$T_0[S_{12}] \oplus T_1[S_1] \oplus T_2[S_6] \oplus T_3[S_{11}] \oplus RK$
$T_0[S_0] \oplus T_1[S_5] \oplus T_2[S_{10}] \oplus T_3[S_{15}] \oplus RK \oplus exp\_twk$
$T_0[S_4] \oplus T_1[S_9] \oplus T_2[S_{14}] \oplus T_3[S_3] \oplus RK \oplus exp\_twk$

Figure 14. Parallel operation process of TweAES-128 or TweAES-128-6 using T-table method.

5. Results

Table 7 is an environment in which the results were measured by applying the methods proposed by us to the ESTATE algorithm using OpenCL parallel processing, the ESTATE algorithm implemented with Web Assembly, and the reference C ESTATE algorithm.

Table 8 is a comparison result of OpenCL parallel processing, AES T-table, extended tweak pre-computation, and ESTATE algorithm applying loop unrolling methods and the reference C code ESTATE algorithm for sequential processing. We measured the process of creating a total of 6,400 ciphertexts and tags, respectively. As a result, in ESTATE TweAES-128, TweAES-128-6 and TweGIFT-128, OpenCL was 6.69 times, 7.31 times, and 1.47 times faster than the reference C/C++ code, respectively.

Table 7. Performance measurement environment.

Operationg System	CPU	RAM	SW	Languages and API	Used Input Value	ESTATE Operation Count
Window 10 Education	Intel i5-8250U 1.6 GHz	8 GB	(1) Chrome (2) FireFox (3) Microsoft Edge	(1) C/C++ (2) Web Assembly (3) OpenCL	Nonce: 25,600-byte AD: 51,200-byte Message: 512,000-byte	6400

Table 8. Performance comparison of OpenCL implementation and C/C++ reference code applying the proposed method (ns: nanosecond).

Algorithm	OpenCL	Reference C/C++	Performance Improvement
ESTATE TweAES-128	19,088,500 ns	127,842,877 ns	6.69 times
ESTATE TweAES-128-6	15,966,333 ns	116,813,270 ns	7.31 times
ESTATE TweGIFT-128	1,958,343,000 ms	2,897,251,400 ns	1.47 times

Table 9 shows the result of comparing the algorithm to which the T-table shuffling method was applied and the algorithm not applied. This is a measurement result of the process of shuffling and calculating the 1024-byte T-table. Due to shuffling, performance overhead occurs because memory must be accessed twice, unlike the method not applied. As a result, ESTATE TweAES-128 and TweAES-128-6 show performance overhead of 7% and 51%, respectively.

Table 9. Performance overhead measurement result through application of T-table shuffling method (ns: nanosecond).

Algorithm	Applied T-Table Shuffling Method	Normal Method	Performance Overhead
ESTATE TweAES-128	20,589,394 ns	19,088,500 ns	7%
ESTATE TweAES-128-6	24,192,899 ns	15,966,333 ns	51%

Table 10 shows how much performance overhead occurs compared to C language by implementing the ESTATE algorithm in Web Assembly to use the edge computing method using Web Assembly. Measurements were made for C and Web Assembly using the same input values. Web Assembly was measured on Chrome, FireFox, and Microsoft Edge. As a result, TweAES-128, TweAES-128-6, and TweGIFT-128 implemented as Web Assembly have 11%, 5%, 22% performance overhead in Chrome, 10%, 2%, 54 in FireFox. It shows % performance overhead, and 9%, 6%, and 17% performance overhead in Microsoft Edge. The reason the performance overhead ratio is different for each web browser is that the rendering engine and JavaScript engine used for each web browser are different. However, in the case of TweAES-128 and TweAES-128-6, the performance overhead is not large, so it can be seen that it is efficient to perform edge computing through a web-based application using Web Assembly.

Table 10. Performance overhead measurement result of ESTATE algorithm using Web Assembly (ns: nanosecond).

Algorithm	Reference C/C++ Code	Web Assembly		
		Chrome (Performance Overhead)	FireFox (Performance Overhead)	Microsoft Edge (Performance Overhead)
ESTATE TweAES-128	127,842,877 ns	142,775,000 ns (11%)	141,000,000 ns (10%)	140,374,999 ns (9%)
ESTATE TweAES-128-6	116,813,270 ns	123,155,001 ns (5%)	120,000,000 ns (2%)	124,045,001 ns (6%)
ESTATE TweGIFT-128	2,897,251,400 ns	3,560,440,001 ns (22%)	4,490,000,000 ns (54%)	3,401,205,000 ns (17%)

6. Conclusions

The existing edge computing method takes over the role of cloud computing services in hardware, such as ARM, AVR, and RISC-V. Therefore, there is a disadvantage of having to implement separately using a function and programming language suitable for each environment used in ARM, AVR, and RISC-V. In this paper, we propose a web-based application edge computing method using Web Assembly in order to use an efficient edge computing method.

1. Implementation of ESTATE algorithm using Web Assembly
   Web Assembly was created to show similar performance to low-level language in a web environment. Cryptographic algorithms using web-based applications can use web-based applications, and can be used without additional modification in PCs, smart phones, and IoT devices used as edge devices. Therefore, even if the platforms used are different, it is also cost-effective because it can be used generally without additional modification in terms of implementation. In addition, web-based application edge computing communicates with various platforms, so, to send data securely, we implement and use the ESTATE algorithm, which has both encryption and authentication processes, in Web Assembly. We can see how Web Assembly has caught up with the performance of low-level languages. ESTATE Web Assembly implementation compares performance with reference C/C++ code. Web Assembly implementation is measured in web browsers Chrome, FireFox, and Microsoft Edge. As a result, TweAES-128, TweAES-128-6, and TweGIFT-128 implemented as Web Assembly have 11%, 5%, 22% performance overhead in Chrome, 10%, 2%, 54 in FireFox. It shows % performance overhead, and 9%, 6%, and 17% performance overhead in Microsoft Edge. As a result, it is slower than C/C++, which is a low-level language, but it can be used efficiently because it can be used without special modifications on devices that can use web-based applications.
2. ESTATE algorithm using OpenCL parallel processing
   Data processed by the web-based application edge computing method are eventually stored on the main server. Therefore, in order to use the ESTATE algorithm efficiently, it is necessary to implement it according to the server environment. So, we propose a method of simultaneously processing ciphertext and tag generation to be sent to multiple platforms using OpenCL parallel processing. Through OpenCL parallel processing, each byte value is processed simultaneously instead of sequentially for the 16-byte input value used for one encryption process. OpenCL has a load when using conditional statements. In the ESTATE algorithm, a conditional statement is used to XOR the extended tweak value every specific round. Therefore, the loop unrolling method was used to remove the performance load by removing the process of using conditional statements. In addition, data is stored in a local memory with a fast operation speed and encrypted to perform efficient operation. For performance comparison, we compare the OpenCL parallel processing implementation and the reference C/C++ sequential processing implementation. As a result, the OpenCL implementation shows about 6.69 times, 7.31 times, and 1.47 times performance improvement in ESTATE TweAES-128, TweAES-128-6, and TweGIFT-128 than the reference C/C++ implementation.
3. Method for efficient and safe operation of ESTATE algorithm
   Additional methods are applied to safely and efficiently operate the ESTATE algorithm itself. The ESTATE algorithm uses conditional statements to check the type of input value to be encrypted, check whether it is the last block, check the tweak value, and calculate the extended tweak value for each specific round. The 8-bit and 32-bit extended tweak values used in TweAES-128, TweAES-128-6, and TweGIFT-128 are stored and used in advance through pre-calculation. This method reduces the performance load by removing unnecessary conditional statements. In addition, TweAES-128 and TweAES-128-6 have the same operation process as the AES algorithm, so they may be vulnerable to cache-timing attacks. So, we apply the T-table shuffling method, which is a previously studied method, to operate safely. We reduced the performance load by applying the proposed methods to minimize the performance load even when the T-table shuffling method is applied. As a result of applying the T-table shuffling method, TweAES-128 and TweAES-128-6 show about

7% and 51% performance overhead, respectively, than those without applying the T-table shuffling method.

4. Future Work
Web-based application using Web Assembly can be used in various devices without additional modification, so it can reduce the system load of the server and is effective in responding to failures. Web Assembly is currently continuously developing, and, since various devices, such as PCs, smart phones, and smart devices, are developing more and more, web technology is also developing accordingly. Currently, technologies using high-end hardware, such as Web Assembly's SIMD technology and WebGPU, are being developed. In addition, it is being developed so that Web Assembly and WebGPU can be used together. When these technologies become stable in the future, many web developers will develop web services using various technologies, such as SIMD and WebGPU. Therefore, it can be used in various ways in terms of crypto security, and various studies will be conducted using web technologies developed in the field of crypto security. Therefore, the web-based application edge computing method can also be developed, and performance will be improved. Currently, there are various NIST LWC (National Institute of Standards and Technology LightWeight Cryptography) Round 2 candidate algorithms. However, the OpenCL parallel processing method we used is a method applicable to other candidate algorithms. Even if the LWC algorithm other than ESTATE is used to send data to multiple devices, the service can be provided more efficiently by using the method of simultaneously processing multiple ciphertexts and tags through the OpenCL parallel processing method.

Author Contributions: Writing—original draft, B.P.; Writing—review and editing, S.C.S. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2019R1F1A1058494).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Hayes, B. *Cloud Computing*; Communications of the ACM: New York, NY, USA, July 2008.
2. Yu, W.; Liang, F.; He, X.; Hatcher, W.G.; Lu, C.; Lin, J.; Yang, X. A Survey on the Edge Computing for the Internet of Things. *IEEE Access* **2018**, *6*, 6900–6919. [CrossRef]
3. Ai, Y.; Peng, M.; Zhang, K. Edge computing technologies for Internet of Things: A primer. *Digit. Commun. Netw.* **2018**, *4*, 77–86. [CrossRef]
4. Wang, X.; Han, Y.; Leung, V.C.M.; Niyato, D.; Yan, X.; Chen, X. Convergence of Edge Computing and Deep Learning: A Comprehensive Survey. *IEEE Commun. Surv. Tutor.* **2020**, *22*, 869–904. [CrossRef]
5. Tilkov, S.; Vinoski, S. Node.js: Using JavaScript to build high-performance network programs. *IEEE Internet Comput.* **2010**, *14*, 80–83. [CrossRef]
6. Park, B.; Song, J.; Seo, S.C. Efficient Implementation of a Crypto Library Using Web Assembly. *Electronics* **2020**, *9*, 1839. [CrossRef]
7. Rossberg, A.; Titzer, B.L.; Haas, A.; Schuff, D.L.; Gohman, D.; Wagner, L.; Zakai, A.; Bastien, J.F.; Holman, M. Bringing the web up to speed with WebAssembly. *Commun. ACM* **2018**, *61*, 107–115. [CrossRef]
8. Rossberg, A. WebAssembly Specification Release 1.1, 2020. Available online: https://webassembly.github.io/spec/core/ (accessed on 25 February 2021).
9. Ritchie, D.M. The development of the C language. *ACM Sigplan Not.* **1993**, *28*, 201–208. [CrossRef]
10. Smith, E. The C++ Language. In *Introduction to the Tools of Scientific Computing*; Springer: Berlin, Germany, 2020; pp. 133–148.
11. Doglio, F. An Introduction to TypeScript. In *Introducing Deno*; Springer: Berlin, Germany, 2020; pp. 27–62.
12. Bhattacharjee, J. Basics of Rust. In *Practical Machine Learning with Rust*; Springer: Berlin, Germany, 2020; pp. 1–30.
13. Sjölander, Erik. Krypteringsalgoritmer i OpenCL: AES-256 och ECC ElGamal, 2012. Available online: https://www.diva-portal.org/smash/get/diva2:555565/FULLTEXT01.pdf (accessed on 25 February 2021).

14. Munshi, A.; Gaster, B.; Mattson, T.G.; Ginsburg, D. *OpenCL Programming Guide*; Pearson Education: London, UK, 2011.
15. Munshi, A. The opencl specification. In Proceedings of the 2009 IEEE Hot Chips 21 Symposium (HCS), Stanford, CA, USA, 26 May 2009; pp. 1–314.
16. Chakraborti, A.; Datta, N.; Jha, A.; Mancillas-López, C.; Nandi, M.; Sasaki, Y. ESTATE: A Lightweight and Low Energy Authenticated Encryption Mode. *IACR Trans. Symmetric Cryptol.* **2020**, *2020*, 350–389. [CrossRef]
17. Black, J.; Rogaway, P. *CBC MACs for Arbitrary-Length Messages: The Three-Key Constructions*; Springer: Berlin, Germany, 2000; pp. 197–215.
18. Dworkin, M. *Recommendation for Block Cipher Modes of Operation. Methods and Techniques*; Technical Report; National Inst of Standards and Technology: Gaithersburg, MD, USA, 2001.
19. Yao, J.; Zimmer, V. Cryptography. In *Building Secure Firmware*; Springer: Berlin, Germany, 2020; pp. 767–823.
20. Heron, S. Advanced encryption standard (AES). *Netw. Secur.* **2009**, *2009*, 8–12. [CrossRef]
21. Banik, S.; Pandey, S.K.; Peyrin, T.; Sasaki, Y.; Sim, S.M.; Todo, Y. GIFT: A Small Present—Towards Reaching the Limit of Lightweight Encryption. In Proceedings of the Cryptographic Hardware and Embedded Systems—CHES 2017—19th International Conference, Taipei, Taiwan, 25–28 September 2017; pp. 321–345. [CrossRef]
22. Sanida, T.; Sideris, A.; Dasygenis, M. Accelerating the AES Algorithm using OpenCL. In Proceedings of the 2020 9th International Conference on Modern Circuits and Systems Technologies (MOCAST), Bremen, Germany, 18 September 2020; pp. 1–4.
23. Yuan, Y.; He, Z.; Gong, Z.; Qiu, W. Acceleration of AES encryption with OpenCL. In Proceedings of the 2014 Ninth Asia Joint Conference on Information Security, Wuhan, China, 29 January 2015; pp. 64–70.
24. Wang, X.; Li, X.; Zou, M.; Zhou, J. AES finalists implementation for GPU and multi-core CPU based on OpenCL. In Proceedings of the 2011 IEEE International Conference on Anti-Counterfeiting, Security and Identification, Xiamen, China, 29 July 2011; pp. 38–42.
25. Conti, V.; Vitabile, S. Design exploration of aes accelerators on fpgas and gpus. *J. Telecommun. Inf. Technol.* **2017**, *1*, 28–38.
26. Di Biagio, A.; Barenghi, A.; Agosta, G.; Pelosi, G. Design of a parallel AES for graphics hardware using the CUDA framework. In Proceedings of the 2009 IEEE International Symposium on Parallel & Distributed Processing, Rome, Italy, 10 July 2009; pp. 1–8.
27. D Amato, J.P.; Vénere, M.J. Encrypting video and image streams using OpenCL code on-demand. *CLEI Electron. J.* **2014**, *17*, 6.
28. Matsui, M. Linear cryptanalysis method for DES cipher. In *Workshop on the Theory and Application of of Cryptographic Techniques*; Springer: Berlin, Germany, 1993; pp. 386–397.
29. Schneier, B. *Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)*; Springer: Berlin, Germany, 1993; pp. 191–204.
30. Rivest, R.L.; Shamir, A.; Adleman, L. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM* **1978**, *21*, 120–126. [CrossRef]
31. Inampudi, G.R.; Shyamala, K.; Ramachandram, S. Parallel implementation of cryptographic algorithm: AES using OpenCL on GPUs. In Proceedings of the 2018 2nd International Conference on Inventive Systems and Control (ICISC), Coimbatore, India, 28 June 2018; pp. 984–988.
32. Standaert, F.X. Introduction to side-channel attacks. In *Secure Integrated Circuits and Systems*; Springer: Berlin, Germany, 2010; pp. 27–42.
33. Roh, D.; Koo, B.; Jung, Y.; Jeong, I.; Lee, D.; Kwon, D.; Kim, W. Revised Version of Block Cipher CHAM. In Proceedings of the Information Security and Cryptology—ICISC 2019—22nd International Conference, Seoul, Korea, 4–6 December 2019; pp. 1–19. [CrossRef]
34. King, B. *wNAF*, an Efficient Left-To-Right Signed Digit Recoding Algorithm*; Springer: Berlin, Germany, 2008; pp. 429–445.
35. Chevallier-Mames, B.; Ciet, M.; Joye, M. Low-cost solutions for preventing simple side-channel analysis: Side-channel atomicity. *IEEE Trans. Comput.* **2004**, *53*, 760–768. [CrossRef]
36. Fips Pub. *Secure Hash Standard (SHS)*; Fips Pub: Edmonds, WA, USA, March 2012; Volume 180.
37. Turner, J.M. The keyed-hash message authentication code (hmac). *Fed. Inf. Process. Stand. Publ.* **2008**, *198*, 1.
38. Rösch, J. Efficient Implementation of Picnic. 23 September 2020. Available online: https://is.muni.cz/th/pbn05/Efficient_implementation_of_Picnic.pdf (accessed on 25 February 2021.)
39. Chase, M.; Derler, D.; Goldfeder, S.; Orlandi, C.; Ramacher, S.; Rechberger, C.; Slamanig, D.; Zaverucha, G. *The Picnic Signature Scheme Design Document (Version 1.0)*; NIST Post-Quantum Cryptogr. Stand. Round; NIST: Gaithersburg, MD, USA, 2017; Volume 3.
40. Daehyeon Bae, J.H.; Ha, J. Implementation of AES Resistant to Cache Side-Channel Attack Using T-Table Shuffling Method. *Conf. Inf. Secur. Cryptogr. Winter* **2020**, *30*, 579–583.
41. Young, E.A.; Hudson, T.J.; Engelschall, R.S. OpenSSL. 9 November 2001. Available online: http://www.openssl.org/ (accessed on 25 February 2021)
42. Bernstein, D.J. Cache-Timing Attacks on AES. 14 April 2005. Available online: https://cr.yp.to/antiforgery/cachetiming-20050414.pdf (accessed on 25 February 2021).
43. Fisher, R.A.; Yates, F. *Statistical Tables: For Biological, Agricultural and Medical Research*; Oliver and Boyd: Edinburgh, UK, 1938.