Future Generation Computer Systems 96 (2019) 336–347

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Energy-efficient crypto acceleration with HW/SW co-design for HTTPS[☆]

Chunhua Xiao[a,*], Lei Zhang[a], Weichen Liu[b], Neil Bergmann[c], Yuhua Xie[a]

[a] School of Computer Science, Chongqing University, Chongqing, China
[b] School of Computer Science and Engineering, Nanyang Technological University, Singapore
[c] School of Information Technology and Electrical Engineering, University of Queensland, Australia

HIGHLIGHTS

- Comprehensive analysis of the energy efficiency for HTTPS crypto process.
- Proposed a request reconstruction scheme to make full utilization of hardware crypto accelerators.
- Proposed an adaptive scheduling strategy to process crypto requests dynamically.
- Design a dynamic management mechanism to make full use of system resource.

ARTICLE INFO

Article history:
Received 16 May 2018
Received in revised form 25 January 2019
Accepted 14 February 2019
Available online 21 February 2019

Keywords:
Energy efficiency
HW/SW co-design
Hardware acceleration
HTTPS
OpenSSL

ABSTRACT

Entering the Big Data era leads to the rapid development of web applications which provide high-performance sensitive access on large cloud data centers. HTTPS has been widely deployed as an extension of HTTP by adding an encryption layer of SSL/TLS protocol for secure communication over the Internet. To accelerate the complex crypto computation, specific acceleration instruction set and hardware accelerator are adopted. However, energy consumption has been ignored in the rush for performance. Actually, energy efficiency has become a challenge with the increasing demands for performance and energy saving in data centers. In this paper, we present the *EECA*, an Energy-Efficient Crypto Acceleration system for HTTPS with OpenSSL. It provides high energy-efficient encryption through HW/SW co-design. The essential idea is to make full use of system resource to exert the superiorities of different crypto acceleration approaches for an energy-efficient design. Experimental results show that, if only do crypto computations with typical encryption algorithm AES-256-CBC, the proposed *EECA* could get up to 1637.13%, 84.82%, and 966.23% PPW (Performance per Watt) improvement comparing with original software encryption, instruction set acceleration and hardware accelerator, respectively. If considering the whole working flow for end-to-end secure HTTPS based on OpenSSL with cipher suite ECDHE-RSA-AES256-SHA384, *EECA* could also improve the energy efficiency by up to 422.26%, 40.14% and 96.05% comparing with the original Web server using software, instruction set and hardware accelerators, respectively.

1. Introduction

A strong tide of constructing green cloud computing data center has engendered with the rapid development of data volume [1]. Continuously developing transaction-intensive Web applications makes the security become increasingly important [2]. More and more transactions require the transfer of sensitive information kept on data center through the Internet. However, the Internet is an insecure medium. Data transmitted from and stored on devices will need to be protected [3,4]. Data security is of great concern especially for enterprises that build their private clouds [5].

Luckily, Hypertext Transfer Protocol Secure (HTTPS) creates a secure channel over an insecure network, with especially wide deployment on the Internet. In HTTPS, the communication protocol is encrypted by Transport Layer Security (TLS), or formerly, its predecessor, Secure Sockets Layer (SSL) [6]. SSL/TLS is the industry's best and most accepted standard cryptographic protocols. In order to protect the security of sensitive data during transmission, more and more governments, enterprises and banks begin to deploy HTTP over SSL/TLS. A large number of user requirements for

☆ This work is supported by National Natural Science Foundation of China: No. 61502061, Chongqing Application Foundation and Research in Cutting-edge Technologies: No. cstc2015jcyjA40016, the Fundamental Research Funds for the Central Universities: No. 106112017CDJXY180004, and also the financial support from the program of China Scholarship Council: No. 201706055029.

* Corresponding author.
E-mail addresses: xiaochunhua@cqu.edu.cn (C. Xiao), zhanglei51@cqu.edu.cn (L. Zhang), wchliu@gmail.com (W. Liu), bergmann@itee.uq.edu.au (N. Bergmann), 20151402021@cqu.edu.cn (Y. Xie).

https://doi.org/10.1016/j.future.2019.02.023
0167-739X/© 2019 Elsevier B.V. All rights reserved.

network transmission efficiency and speed of response higher and higher.

The most widely deployed, freely available implementation of SSL/TLS protocol is the OpenSSL [7]. The core library of OpenSSL implements basic cryptographic functions and various utility functions [8]. The cryptographic functions, such as symmetric key ciphers, are extremely compute-intensive. OpenSSL does these expensive computations through software implementations. It may not be able to compete with the increasing need for performance of encryption services.

In order to improve the performance of crypto operations, two approaches are exploited. One is the new instruction set which can handle complex steps of cryptographic algorithms, such as the Intel Advanced Encryption Standard New Instructions (AES-NI) [9]. The other is the special hardware accelerator which is able to offload the computation from CPU, such as the Intel QuickAssist Technology (QAT) cards [10].

Unfortunately, for secure communication with SSL/TLS, software encryption is not so energy-efficient as instruction set or hardware accelerator. We find that both instruction set and hardware accelerator have limited capacity in serving crypto requests of diversity data sizes. On the one hand, the software and instruction set consumes many CPU cycles when doing crypto operations. The CPU resource can become the bottleneck for performance improvement. On the other hand, when the data size is small, the overhead of hardware initialization and interrupt processing neutralizes the benefit of hardware acceleration and CPU offloading. Why not take full advantages of these solutions in the same system to select the most suitable encryption mechanism adaptively?

Furthermore, trends such as cloud computing and big data applications require huge data centers with thousands of servers as well as a storage and communication network infrastructure [11,12]. A 2016 report forecasted that data center workload will more than double between 2015 and 2020, while the number of servers installed is expected to grow by 25 percent [13,14]. This rapid growth has been accompanied by increasing energy consumption. Energy consumption has been widely studied for many years in the computer engineering community, focusing on designing processors that consume very little energy using Dynamic Voltage Frequency Scaling (DVFS) and on some other power saving techniques [15]. Regarding server, green computing has been introduced as a field that studies ways to address software solutions from a green, sustainable, and energy efficient perspective. Based on these works, we can observe how there is a trend in building energy efficient Web server [16–18]. However, deploying security mechanisms for Web server and HTTPS services consumes a huge amount of power. During a secure session, the main sources of energy consumption are session establishment, packet transmission and reception, and cryptographic computation. For encrypting smaller data packets, ECC-3DES-SHA is more energy efficient, whereas for encrypting bigger data packets, RSA-RC5-SHA consumes less energy [19]. Web server is rapidly being throttled by the constraints of power delivery and cooling, generating an energy wall for high-density servers that seek to maximize commercial server encryption capacity [20]. Many companies are also starting to be concerned with the energy consumption of their computations [21]. Thus, the issue of energy efficiency is becoming an emerging challenge growing with respect to technical, financial, and environmental reasons. As Luiz Andr Barroso said [22], developing computing infrastructure, the focus should be broader than performance alone. Breaking through this energy wall requires a focus on Performance per Watt instead of absolute performance [23].

In this paper, we propose *EECA*, an Energy-Efficient Crypto Acceleration system for HTTPS with OpenSSL. The goal of this system is to provide high energy-efficient encryption through HW/SW co-design. The goals of this scheme are two folds: boost the Web server energy efficiency and improve system resource utilization.

The essential idea is making full utilization of system resource to exert respective superiorities of instruction set and hardware accelerator for an energy-efficient design. We present *Request Reconstruction* scheme to aggregate SSL/TLS fragments and submit the entire data to the hardware for encryption, so as to give full play to the hardware accelerator. Besides, we proposed a *Dynamic Management Mechanism* to choose the most energy-efficient methodology for crypto acceleration. Our contributions are concluded as below:

- *Comprehensive analysis of the energy efficiency for HTTPS crypto process.* We break down the working flow of secure HTTPS connection through SSL/TLS in detail, and find out the most important factors affecting the performance and energy consumption. Then we explore and analyze the energy efficiency of the existing crypto computation methodologies, including the original software encryption through OpenSSL lib, instruction acceleration and hardware accelerator. Based on the comprehensive analysis, we concluded the pros and cons of different crypto techniques, which paves the way for HW/SW co-design.
- *Proposed a request reconstruction scheme to make full utilization of hardware crypto accelerators.* We found the crypto accelerators are deficiency for existing working flow, which invoke accelerators frequently for small data blocks. The proposed request reconstruction scheme is able to aggregate the fragments to one large data block before encryption, which helps reduce the invocation overhead greatly, and improve the energy efficiency.
- *Proposed an adaptive scheduling strategy to process crypto requests dynamically.* To get the best performance per watt, we adopt a dynamic request scheduler to choose the optimal encryption approach, according to the request characters and the utilization of system resource. It is able to exert respective superiorities of both software/instruction set encryption and hardware crypto computation.
- *Design a dynamic management mechanism to make full use of system resource.* Besides exerting respective superiorities of each encryption approaches by the request scheduler, our designed dynamic management mechanism is able to take full utilization of both software/instruction set and hardware accelerator according to the utilization of hardware and CPU to boost crypto energy efficiency.

The rest of this paper is organized as follows. Section 2 presents the background and the challenges of the existing encryption approaches for secure HTTPS connection with based on SSL/TLS. The proposed design methodology of *EECA* is described in Section 3. Our experimental evaluation and comparisons are illustrated in Section 4. Related work is discussed in Section 5, and we concluded our work in Section 6.

2. Background & motivation

The proposed technique aims to improve the energy efficiency of sensitive data transactions with HTTPS based on OpenSSL. In this section, we first briefly describe the operation mechanism of the HTTP over SSL/TLS. Then, we discuss the challenges of existing crypto-acceleration approaches on the energy efficiency.

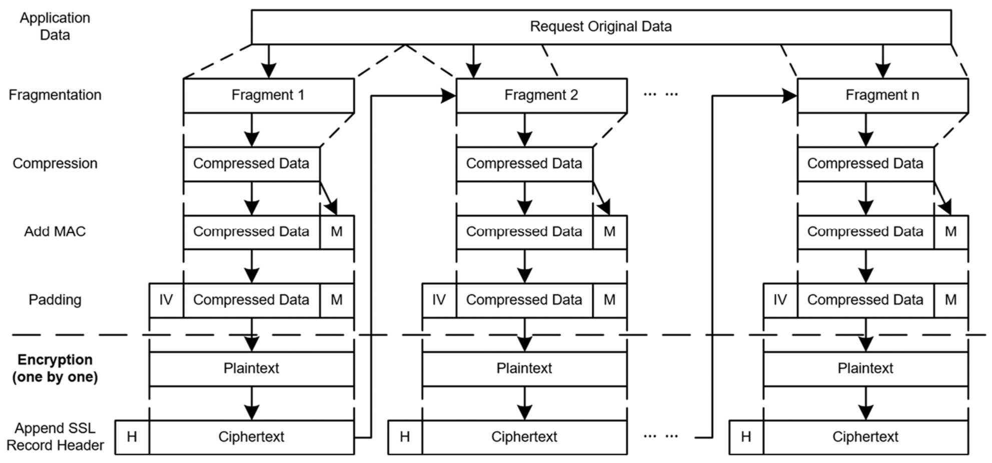


Fig. 1. SSL Record Protocol Process.

2.1. *HTTP over SSL/TLS*

HTTPS (Hypertext Transfer Protocol Secure) is an internet communication protocol that protects the integrity and confidentiality of data between clients and servers [24]. Clients expect a secure and private online experience when connecting with servers. Data transmission using HTTPS is secured via Transport Layer Security protocol (TLS) or its predecessor, Secure Sockets Layer (SSL). It allows clients and servers to communicate in a secure manner over a TCP/IP network. SSL/TLS provides data encryption, server authentication, message integrity and optional client authentication for TCP/IP connections. Transmitted data is encrypted and hidden, so that communications between client/server applications are not eavesdropped by attackers and the data is not changed during transmission, that is, to ensure the integrity of data. The identity certificate is used to prevent identity impersonate.

SSL/TLS is not a single protocol but rather two main sublayers of protocols. One is the higher-layer protocols including the Handshake Protocol, the Change Cipher Spec Protocol, the Alert Protocol and the Application Data Protocol, the other is the Record Protocol. The Handshake Protocol is used before any application data is transmitted. When a client and a server first start SSL/TLS communication after establishing the TCP connection they agree on a protocol version, select cryptographic algorithms, optionally authenticate each other, and use public-key encryption techniques to generate shared secrets. After the SSL/TLS session established, the Record Protocol takes messages to be transmitted, fragments the data into manageable blocks of 16 kB or less, optionally compresses the data, applies a MAC, encrypts, and transmits the result by TCP packet. When the SSL/TLS session is ending, both the client and the server close the TCP socket.

Fig. 1 shows the original SSL/TLS Record Protocol process. A request data crypto operation is divided into six steps:

Step 1. Each upper-layer original data is fragmented into blocks of 16 kB or less.

Step 2. Each fragment is compressed in order to save Internet bandwidth when transmitting. This step is optional and it requires extra computation power to carry out compression.

Step 3. The compressed data is authenticated with a message authentication code (MAC). For this purpose, a shared secret key is used. In essence, the hash code is calculated over a combination of the message, a secret key, and some padding.

Step 4. Padding bits are added to the buffer so that the size meets the requirement of encryption algorithms.

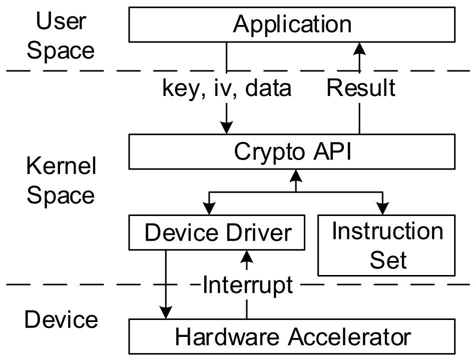


Fig. 2. Working flow of instruction set and hardware accelerator.

Step 5. The data buffer is encrypted one by one. Note that the size of encryption request is around 16 kB, if hardware accelerator is used, there is a large overhead for hardware management, such as interrupt handling and hardware initialization.

Step 6. SSL header is appended to the ciphertext and the buffer is sent to the client via the network.

2.2. *Cryptographic working flow*

The cryptographic operations are transparent to the user space applications. However, the cryptographic operations in SSL/TLS are time-consuming, in order to accelerate the execution of encryption algorithm, the new instruction set, such as Intel Advanced Encryption Standard New Instructions (AES-NI) [9], and the special hardware accelerator, such as the Intel QuickAssist Technology (QAT) cards [10], are designed to implement some of the complex and performance intensive steps of cryptographic algorithms.

Fig. 2 shows the working flow of instruction set and hardware accelerator. The working flow includes 5 basic parts: application, Crypto API, instruction set, device driver and hardware accelerator. For encryption requests, the application issues encryption system call to pass the key, IV, and source data address to the kernel Crypto API. Then, Crypto API passes requests to hardware accelerator through device driver or directly uses the instruction set to finish the encryption. Crypto API acts as the middle layer between application and hardware accelerator. The Crypto API interacts with the upper-layer application, receives delivered require data and returns completed cryptograph. Downwards, Crypto API transforms the received request as the data structure which can be identified by the device driver. Device driver performs 4 major works for accelerator invocation: initialization of hardware engines, device loading and unloading, encryption algorithm registration and interrupt processing. The driver provides API for the kernel to receive delivered data from Crypto API and constructs them as the data block that hardware accelerator can resolve, then delivers prepared data to hardware engines. Once interrupt signal from hardware accelerator is detected, the driver will get encryption results and return to the upper layer for further transmission.

2.3. Motivation

In order to optimize the energy efficiency of Web server, we first investigate the working flow of the secure HTTPS connection for Web server and present the time breakdown of the major steps. Then, we discuss how the existing acceleration approaches bear limited capacity in providing energy-efficient Web server system.

2.3.1. Breakdown of HTTPS working flow

We use Nginx, a popular Web server, with ECDHE-RSA-AES256-SHA cipher suite, to analyze the working flow quantitatively. We divide the working flow into five parts: handshake, read file, crypto, network and others, as discussed in Section 2.1.

Fig. 3 shows the time breakdown of Nginx for various pages. As the experiment results illustrated in Fig. 3(a), the time of handshake decreases as the page size grows. This is because handshake happens only once for a requested Web page, the RSA algorithm is time-consuming and takes more than half of the time for small pages (less than 32 kB). Besides, the time of crypto increases as the page size grows. This is reasonable in that the large the pages, more cryptographic operations are needed to encrypt the contents.

If we enable the keep alive option, an SSL/TLS connection can be reused for different Web pages. The results in the keep-alive working mode are showed in Fig. 3(b). As we can see, the time for handshake drops dramatically, while the time for crypto and network increases with keep alive. The crypto time increases from 40.47% to 62.82% for 2 MB pages.

Based on the above results, it is clear that the crypto operations are time-consuming. Thus, in order to optimize the energy consumption of Web server, the crypto operations must be energy-efficient as much as possible. The energy efficiency for sending packages in the network is hard to optimize since the network condition cannot be controlled easily.

Fortunately, new acceleration approaches, such as instruction set and special hardware accelerators are developed to achieve this goal. In the next section, we discuss the state-of-the-art encryption approaches behave in terms of performance and energy efficiency.

2.3.2. Challenges of existing crypto-acceleration approaches

In addition to software encryption, instruction set and hardware accelerator are two main alternative methods to achieve crypto operations for SSL/TLS. These approaches mainly devote to improving the encryption performance. However, one fundamental property needs to be researched if the approaches should be deployed in large data centers: How is the energy efficiency of these approaches?

To explore the energy efficiency of state-of-the-art encryption approaches, we run OpenSSL *speed* benchmark with algorithm AES-256-CBC on an 8-Core HUAWEI Taishan server connected with a digital power meter. The server includes ARM Cortex-A57 CPUs which support the ARMv8 Cryptography Extensions. The Cryptography Extensions have new instructions that can be used to accelerate the execution of AES, SHA1, and SHA2-256 algorithms, which implement similar functionality as Intel AES-NI. Besides, the HUAWEI Taishan server already includes hardware accelerators in its own SoC.

As shown in Fig. 4, we use Performance per Watt (PPW) to denote the energy efficiency for encryption methods, which is defined by the maximum performance divided by its energy consumption. The experimental results reveal problems of the existing encryption approaches.

- For both encryption PPW and energy consumption, software implementation is inferior to other two alternative methods.
- For small data size, hardware accelerator takes the highest energy consumption but achieves lower PPW.
- For large data size, instruction set spends more energy than hardware accelerator but gets lower PPW.
- Software and instruction set almost take up all the CPU resource. The limited CPU resource becomes the bottleneck for further PPW improvement for software and instruction set. While hardware accelerator can significantly release CPU resource by offloading computing-intensive operations.

The reasons that lead to the above results are two folds. First, the granularity of encryption is too small to take full advantages of hardware accelerator. SSL/TLS uses 16 kB or less as the encryption size. Thus, a large request data is split into multiple 16 kB small fragments. However, it is inefficient to call hardware accelerator for encryption offload, since the overhead of invoking hardware accelerator can offset the offloading benefit. Small data blocks result in many interrupts and the CPU has to spend more time managing hardware. Second, when the system is available with both instruction set and hardware accelerator, only one encryption method can be utilized by OpenSSL, regardless of the data size. This is because the kernel uses the encryption approach with the highest priority when serving encryption requests by default, the priority of hardware accelerator is higher than instruction set.

Based on the above analysis, we argue that in order to maximize the energy efficiency of Web server, the following design choices should be considered. First, the appropriate approach to implement the encryption is determined by the size of request data. instruction set is efficient for small blocks while hardware accelerator is efficient for large data blocks. Second, fragments of large encryption request have to be aggregated to take the most use of hardware accelerator, since it is more energy efficient for large data blocks. Third, considering the underused CPU resource in hardware case, we should enhance their PPW to increase the energy efficiency for the whole system.

3. Related work

In this section, we discuss some works that are closely related to ours.

(1) **Encryption algorithm optimization with HW-SW co-design**
Encryption algorithm acceleration with hardware and software has been considered as an effective method for performance optimization [25–31]. Most of these schemes focus on optimizing the operations of crypto algorithms through hardware–software implementations on SoC or PSoC platforms. The AES-ECC hybrid crypto-system in [25] use a hardware–software co-design approach where AES runs on a NIOS II soft-core processor and ECC's scalar multiplication is accelerated by a hardware crypto-processor. This work optimized both AES MixColumn/InvMiColumn operation and ECC Point Addition/Doubling layer. The PSoC hybrid RSA-AES crypto-system in [27] implements AES on FPGA. It also implements RSA and the KeyExpansion of AES in software which runs on MicroBlaze processor available on Xilinx FPGA circuit. This work optimized the hardware-implemented operations for SubBytes and MixColumns of

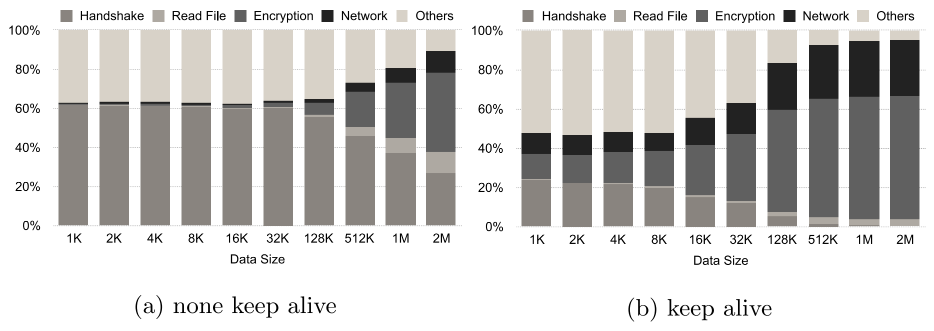


Fig. 3. The time breakdown of Nginx.

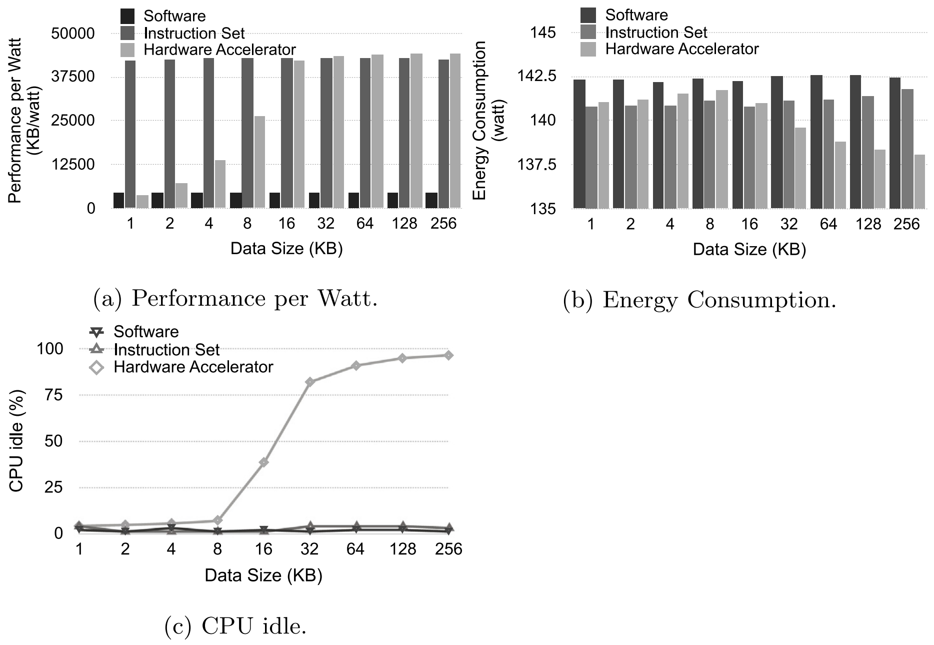


Fig. 4. Energy efficiency of different encryption methods with algorithm AES-256-CBC.

AES. The hardware–software co-design of AES in [29] implements the computationally intensive operations of AES in hardware for better performance. This work also designs the sub-byte calculation with MicroBlaze, a soft-core processor from Xilinx. There is no doubt that these works make effective efforts for the acceleration in encryption. However, they concentrated more on the algorithm implementation itself. In contrast, our work refers how to utilize software/instruction set and hardware accelerators efficiently with higher energy efficiency.

(2) **Energy consumption optimization for web server**
There are some works aim to optimize energy consumption of Web server [32–38]. Dynamic Web-Server (DWS) architecture [32] control server allocation and the routing of requests to selected servers through a reconfigurable switching unfractured. It promotes energy efficiency of Internet server clusters by balancing the cost of energy against the performance. The web server load balancer in [36] is able to distribute requests in a power-efficient manner. This work applied the strategies to a web server environment by altering the balancing schemes to achieve an energy-efficient usage of the nodes within the server farm. The distributed dynamic voltage scaling (DVS) control algorithm in [38] minimizes overall power consumption in a server pipeline subject to end-to-end latency constraints. This work explored the benefits of DVS for power management in server farms. However, this work solved the problem from the hardware viewpoint, and very complex and not easy to apply in the real application. MAS-SJ [39] optimally distributed power usage on both spoofing and jamming attacks by applying dynamic programming. Our scheme differs with other schemes is that we make the breakdown for the major steps of processing HTTPS request by web server and found out the energy-intensive process. We are more focused on optimizing the energy efficiency of these stages. The design concept of our proposed *EECA* can be deployed in other encryption systems if instruction set or hardware accelerator is present in the system.

4. *EECA* design

Based on the analysis in Section 2.3, we find that these two encryption methods bear both pros and cons. Instruction set has greater PPW and lower energy consumption than software encryption, but consumes too many CPU cycles the same as software. Hardware accelerator is able to offload a large portion of CPU burden and get higher PPW with lowest energy consumption only for large data size. Therefore, we proposed *EECA*, an Energy-Efficient Crypto Acceleration system for HTTPS through HW/SW co-design. In this section, we first present the design overview of *EECA* System. Then we will discuss the design decisions in details.

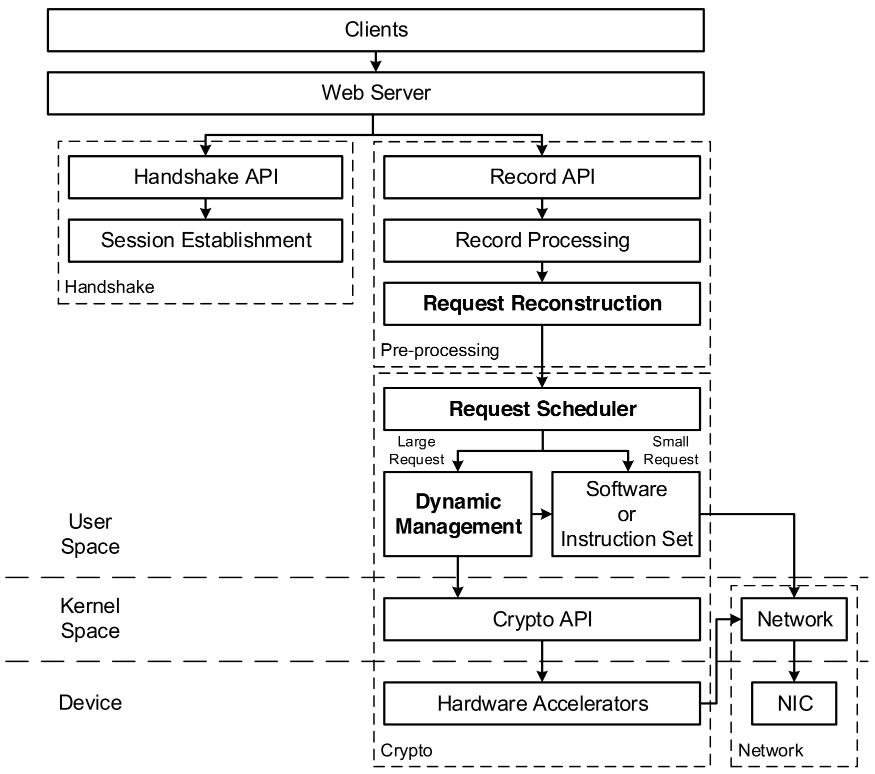


Fig. 5. Architecture overview of *EECA*.

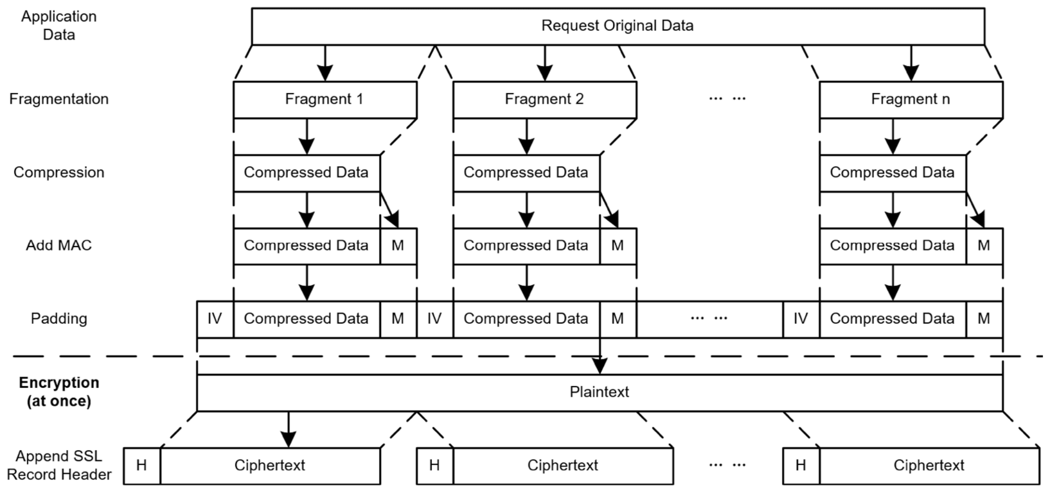


Fig. 6. SSL record protocol operation with request reconstruction.

4.1. Overview

The design of *EECA* is based on three observations.

First, hardware accelerator has advantages in large data encryption, but as discussed in Sections 2.1 and 2.2, the Record Protocol first fragments the data into manageable blocks of 16 kB or less, then submit these fragments to hardware for encryption. Each time the hardware is called, it will generate Mode Switch and Context Switch. Therefore, fragments of one request should be reconstructed to do hardware encryption only one time.

Second, as discussed in Section 2.3, it is obvious that no matter what size the data is, instruction set implementation has a stable and nice energy efficiency. Nevertheless, for large data blocks, hardware implementation has an edge over instruction set implementation. For small data blocks, the overhead of hardware initialization and interrupt processing neutralizes the benefit of hardware acceleration. For higher energy efficiency, encryption requests should be dispatched to appropriate method according to the data size.

Third, as discussed in Section 2.3, each encryption task implemented by instruction set occupies one CPU. Hardware accelerator offloads compute-intense operations and frees up CPUs. The idle resource in hardware case should be utilized to boost the system performance even further and reduce the cost of energy consumption.

Fig. 5 shows the architecture overview of *EECA*. It consists of four parts: handshake, pre-processing, crypto and network. Before any application data is transmitted, client and Web server need to establish a session through the handshake step. As discussed in Section 2.3, the overhead of handshake step can be reduced by using keep alive. In the pre-processing step, the plaintext is first fragmented if the data size is larger than 16 kB. Then each fragment is processed with regular SSL/TLS processing before encryption. Then the requests are reconstructed by their respective SSL/TLS processed fragments. After that each request is scheduled to be encrypted at once in the crypto step. Encryption requests are dynamically managed between instruction set and hardware. Finally, the ciphertext is encapsulated in TCP packets according to the specified data size and transmitted through the Internet. We make the following design decisions.

(1) *Encrypt a request data at once no matter how large it is.* Encryption fragments have to be aggregated to take the most of hardware accelerator, since hardware accelerator is more energy-efficient for large data blocks. However, each invocation to the hardware accelerator generates twice Mode Switch and thrice Context Switch. In order to reduce these overheads, a request is reconstructed by all its SSL processed fragments before encryption and is encrypted at once. Appropriate encryption methods can be determined according to the request data size and system resource state.
(2) *Schedule encryption requests according to the request data size.* As discussed in Section 2.3, instruction set is energy-efficient for small blocks while hardware accelerator is energy-efficient for large data size. In order to take full energy-efficient advantages of different encryption methods, request scheduler is responsible for choosing appropriate encryption methods. For small requests, instruction set is used. For large blocks, hardware accelerator is used.
(3) *Manage encryption requests according to the hardware accelerators utilization and CPU idle.* As discussed in Section 2.3, considering the underused CPU resource in hardware case, if the hardware accelerators are fully loaded, requests data can be dynamically encrypted using instruction set.

4.2. Request reconstruction

As discussed in Sections 2.1 and 2.2, one of the bottlenecks of encryption is the data fragmentation of SSL/TLS Record Protocol and the overhead of calling hardware accelerator. Thus, it is necessary to reconstruct encryption request in order to reduce overhead of calling hardware accelerator. In this section, we discuss in detail how request reconstruction works.

The problem of the original SSL/TLS Record Protocol process (as shown in Fig. 1) is that the encryption fragments are too small to fully utilize hardware accelerator. It is impossible to get the best hardware performance even with multiple concurrent encryption requests. In order to reduce offloading overhead and take full advantage of hardware accelerator. Only Step 5 (as discussed in Section 2.1) can be optimized. This is because the fragment size is defined in SSL/TLS protocol and cannot be enlarged arbitrarily.

Fig. 6 shows the working flow of the proposed *Request Reconstruction*. The essential idea is to aggregate fragments for encryption and split ciphertext for network transmission. It major differences with the original SSL/TLS record process are listed as follows:

(1) In Step 5 (as discussed in Section 2.1), after padding, all the data buffers are aggregated to form a large encryption request. The data buffers appear in the fragmentation order.
(2) The large buffer is encrypted once instead of many times. If hardware is used for encryption, only one-time initialization and interrupt is required to handle the whole request.
(3) The individual ciphertext is extracted from the resulting buffer and SSL header is pre-appended to each ciphertext. It is impossible to send the whole ciphertext at once, because the SSL protocol defines the maximum packet size for each SSL record. *Request Reconstruction* can make sure that the data received by clients is able to be decrypted correctly.

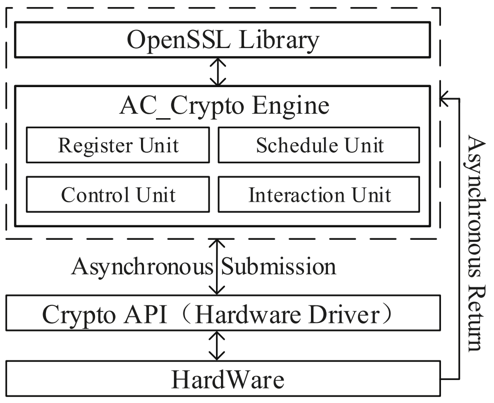


Fig. 7. The scheme of *Adaptive Control Crypto* engine.

The proposed *Request Reconstruction* can greatly improve the performance of hardware accelerator. The optimized hardware encryption also requires cooperation of instruction set and CPU resource. Thus, efficient *Adaptive Control Crypto* engine is critical to achieve great energy efficiency. In the following sections, we describe other design concepts in detail.

4.3. Adaptive control crypto engine

Nowadays, using OpenSSL to implement SSL/TLS is economical, efficient and easy to manage [40]. OpenSSL itself implements basic software library for cryptographic functions and provides various utility functions [8]. When compiling OpenSSL, we can choose whether to use the instruction set function via the configuration options *./configure [no-asm]*. Besides, OpenSSL supports *ENGINE* mechanism since version 0.9.6 [41]. The *ENGINE* framework extends OpenSSL by providing the ability to add various hardware encryption devices to the system. It makes hardware accelerators more transparent and easy to be used.

However, the conventional *EGNINE* framework can only support either hardware or software/instruction set. To make OpenSSL compatibly work with both hardware algorithms and instruction set algorithms at the same time, we design an *Adaptive Control Crypto* engine named *ac_crypto* based on traditional *ENGINE* mechanism. Fig. 7 shows the scheme of the *Adaptive Control Crypto* engine. To implement HW/SW co-design, the *ac_crypto* engine needs to contain the following components.

(1) *Register unit.* It registers the third-party algorithms to OpenSSL protocol library, such as *ac_crypto_aes_cbc*, *ac_crypto_aes_192_cbc* and *ac_crypto_aes_256_cbc*. After loading *ac_crypto* engine, the newly added algorithms will replace the default encryption algorithm of OpenSSL.
(2) *Scheduler unit.* Through a function pointer, OpenSSL gets the interface of instruction set and expands it to the process of hardware implementation. Therefore, *ac_crypto* engine supports instruction set encryption and hardware encryption in the meantime. In order to take full energy efficient advantages of different encryption methods and make the most use of system resource, *scheduler unit* dynamically manages encrypted tasks between hardware and instruction set.
(3) *Control unit.* The parameters of encryption, such as IV, key and source data address, are encapsulated into the data structure which saves the context of the engine. The data object from OpenSSL needs to be processed for supporting hardware algorithm. If the hardware encryption function returns error message, *control unit* will call the instruction set algorithm to encrypt the data. *Control unit* makes the system run in a safe and stable state.

(4) *Interaction unit.* It receives requests from OpenSSL and transmits them to Crypto API, which is the standard encryption frame for dealing with various encryption operations in Linux kernel.

In order to ensure hardware accelerators for normal use when encryption requests arrive at Crypto API, hardware driver must register corresponding encryption algorithm with the highest priority to Crypto API first. Hardware encryption is based on asynchronous mechanism. Processes submit encryption requests to hardware device in an asynchronous manner, and then processes enter the sleeping state. After finishing the operation of encryption, hardware accelerator wakes up the correspondingly sleeping process by interruption and returns the result asynchronously. Therefore, when OpenSSL encrypts large data blocks with hardware, the system has a lot of idle CPU resource.

The essential part of *ac_crypto* engine is the schedule unit, which dispatches requests with a *Dynamic Management Mechanism* according to the request size and system resource. In the next section, we discuss in detail how the *Dynamic Management Mechanism* works.

4.4. Dynamic management mechanism

In *EECA*, we propose *RequestAllocation* algorithm and *DynamicScheduler* algorithm to implement the *Dynamic Management Mechanism* for *schedule unit* in *ac_crypto* engine. From the previous analysis in Section 2.3, we can know that, for large data blocks, hardware encryption with accelerating device can get better PPW and has the advantages of energy consumption comparing with software and instruction set encryption. But when the data size is small, even software encryption has advantages over hardware. In *EECA*, OpenSSL and engine run in user space. To guarantee the universality, the system adopts standard Crypto API framework in kernel space for the invocation of hardware, which includes the overhead of mode switch and context switch. The overhead of hardware implementation is inevitable, and neutralizes the benefit of hardware acceleration for small data blocks. Besides, as to instruction set, the encryption function is accelerated directly through instructions without additional operations needed. Thus, for small data blocks, the performance of hardware is inferior to instruction set.

Algorithm 1: *RequestAllocation*

```
Input: Req: Encrypted requests from applications to OpenSSL.
Output: HW_Queue: request queue encrypted by hardware.
        IS_Queue: request queue encrypted by instruction set.
Get Req from the upper-level applications;
Let Size ← get_request_size(Req);
if size > T then
    Add Req to HW_Queue;
else
    Add Req to IS_Queue;
end
```

In order to bring respective superiorities of instruction set and hardware accelerators into full play, we set up a threshold policy to dynamically dispatch requests between instruction set and hardware. As shown in Algorithm 1, when the upper application submits a new encryption request to OpenSSL, *RequestAllocation* obtains the size of plaintext first. If the data size is greater than the given threshold *T*, this request will be pushed into hardware encryption queue *HW_Queue*, otherwise it will be assigned to instruction set queue *IS_Queue*.

Algorithm 2: *DynamicScheduler*

```
Input: HW_Queue: original queue encrypted by hardware.
       IS_Queue: original queue encrypted by instruction set.
Output: HW_Queue: queue after scheduling encrypted by
        hardware. IS_Queue: queue after scheduling encrypted
        by instruction set.
while 1 do
    if hardware utilization is full then
        if HW_Queue not empty then
            Req ← get_hw_queue_req();
            migrate Req from HW_Queue to IS_Queue;
        end
    else
        if IS_Queue not empty then
            Req ← get_is_queue_req();
            Size ← get_request_size(Req);
            if size > T then
                migrate Req from IS_Queue to HW_Queue;
            end
        end
    end
end
```

Each request in *IS_Queue* needs one CPU to implement encryption. However, requests in *HW_Queue* have low utilization rate of CPUs. In order to enhance the energy efficiency of these free CPUs in hardware case, we propose *DynamicScheduler* algorithm dynamically managing requests between *HW_Queue* and *IS_Queue* according to the utilization of hardware accelerators to maximize the use of system resource.

EECA adopts a background daemon to continuously monitor the utilization rate of hardware device. Algorithm 2 shows how this scheme works. If hardware resource has been fully used and there are still some requests in *HW_Queue*, *DynamicScheduler* will migrate requests from *HW_Queue* to *IS_Queue*. Otherwise, *DynamicScheduler* will transplant request, whose data size is greater than the given threshold *T*, from *IS_Queue* to *HW_Queue*. *DynamicScheduler* can make sure that hardware accelerators be fully used first, and then make the rest of CPU resource bring more PPW with instruction set for OpenSSL.

5. Evaluation

In this section, we evaluate the energy efficiency of the proposed *EECA*. We first present experimental setup and then evaluate *EECA* with benchmarks.

5.1. Experimental setup

To reflect the real working environments satisfying the high concurrent requirements from mass clients, the test platform for the experiments are established with 4 Network Interface Controller, and each contributes 10 Gbps network bandwidth. The experiments are conducted on two 8-Core HUAWEI Taishan servers, with ARM Cortex-A57 CPU running at 2.10 GHz and hardware accelerators in SoC. These ARM Cortex-A57 CPUs support the ARMv8 Cryptography Extensions. The Cryptography Extensions have new instructions that can be used to accelerate the execution of AES, SHA1, and SHA2-256 algorithms. The system is equipped with 128 GB memory. The operating system is adopted as Linux-4.1.27.

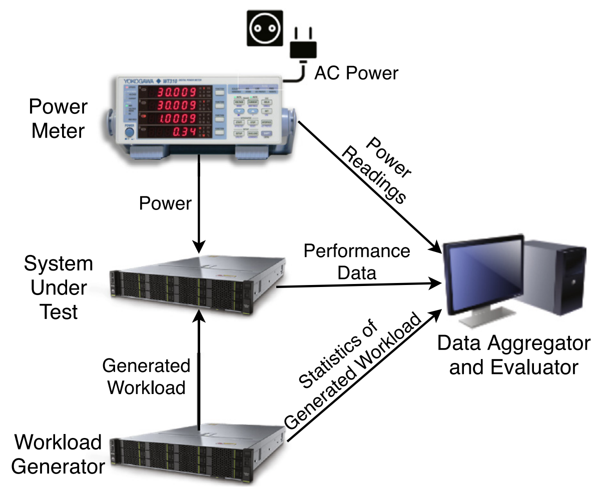


Fig. 8. Overview of energy measurement environment.

As shown in Fig. 8, the energy measurement environment consists of several hardware components that are necessary to perform the test. The following components work together to collect energy consumption and performance data of a Web server by exercising the System Under Test with different benchmark workloads.

One of the two servers is used as a Web server to response HTTPS accesses, that is the System Under Test whose induced energy consumption and performance data will be collected and evaluated by the Data Aggregator and Evaluator. We adopt Nginx 1.11.6 as the HTTP server, which uses an asynchronous event-driven approach to handle requests. OpenSSL-1.0.2j is utilized to perform software encryption and instruction set encryption through the cryptographic library *libcrypto*. All the hardware cryptographic operations are executed through the SoC hardware accelerator. *RequestAllocation* and *DynamicScheduler* are also integrated into OpenSSL with the *ac_crypto* engine for efficient utilization of instruction set and hardware accelerator. In this paper, we set the threshold T to 8 kB based on the above experimental results in Section 2.3. The other server is utilized as a client that access an interactive, transaction-based application on the remote Web server to generate the statistically reproducible workloads that are applied to the System Under Test, that is the Workload Generator.

In this paper, we use YOKOGAWA WT310E digital power meter [42] as the Power Meter, whose readings and setting can be accessed and changed remotely by the Data Evaluator and Aggregator via software WTViewerFreePlus. This Power Meter accepts two input parameters: voltage and current; the accuracy is $\pm 0.1\%$ of reading $+0.05\%$ of range [43]. Moreover, it is only connected to the System Under Test and get the maximum active power every third of a second. We keep the temperature in the server room constant.

5.2. Experimental benchmarks

5.2.1. Web server benchmark

ApacheBench (*ab*) is a command line computer program for benchmarking the HTTPS server [44]. It can accept a single URL, and then repeatedly load the specified multiple independent threads, and use different command line parameters to control the number of visits, the maximum number of concurrent access and so on. At the end of testing, *ab* outputs detailed reports including how many requests per second (RPS) the Web server is capable of serving.

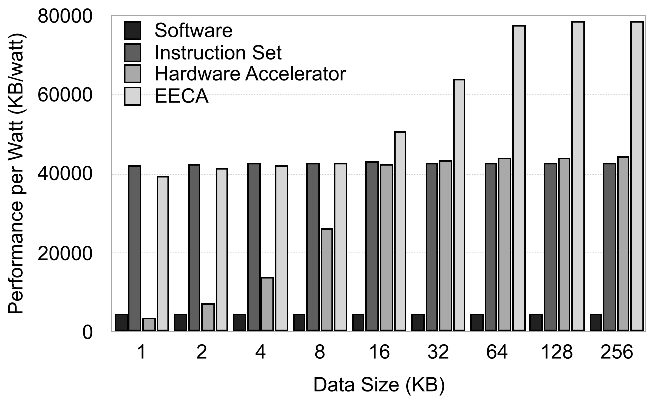


Fig. 9. Energy efficiency of 4 different encryption methods with algorithm AES-256-CBC.

5.2.2. Encryption benchmark

Benchmark *speed*, a part of OpenSSL package, runs for the encryption performance tests. It measures the throughput on various ciphers in terms of the number of bytes processed in a time unit. The OpenSSL *speed* command, by default, runs through every single algorithm in every single supported mode and option, with several different sizes of data, for 3 s [45]. At the end of testing, *speed* reports the encryption performance results in 1000 s of bytes per second (kB/s) processed. Before the experiment, in order to get steady and multiple results of energy consumption and performance, we control the testing time from 3 to 10 s by modifying the *SECONDES* in *OpenSSL-1.0.2j/apps/speed.c* and add several different data size cases.

5.3. Encryption energy efficiency

We first use *speed* to evaluate the energy efficiency of OpenSSL layer with different encryption methods to perform algorithm AES-256-CBC. We record the throughput (TP) and energy consumption (Power) of different encryption approaches in the condition of variable data size, including 1 kB, 2 kB, 4 kB, 8 kB, 16 kB, 32 kB, 64 kB, 128 kB and 256 kB, and divide the performance by its corresponding energy consumption to get the PPW. Fig. 9 shows the energy efficiency of four different encryption methods, including software, instruction set, hardware and *EECA*. To better present the OpenSSL energy-efficiency improvement with our proposed *EECA*, we show the incremental multiples of PPW for *EECA* (Imprv.) comparing with software, instruction set and hardware in Table 1.

As shown in Fig. 9, when data size is smaller than 8 kB, the PPW of *EECA* gradually increased from 39 441.74 kB/W to 42 612.48 kB/W. Furthermore, when data size is larger than 8 kB, the PPW of *EECA* increased from 50 668.60 kB/W to 78 665.05 kB/W. Regardless of data size, *EECA* makes the utmost of CPU and hardware accelerators resource.

(1) ***EECA* vs. software encryption**
The performance of software encryption is poor, because it has no accelerating measures. As shown in Table 1, for different data size, the throughput of *EECA* remarkably outperforms software method. *EECA* gets 766.85% to 1637.13% PPW improvement comparing with software implementation.

(2) ***EECA* vs. instruction set encryption**
When data size is larger than 8 kB, the throughput of *EECA* outperforms instruction set. *EECA* improves 18.02% to 84.82% PPW over instruction set. Beyond that, the throughput and PPW of *EECA* is a little bit lower than instruction set. The reason is that a fraction of CPU resource is occupied for data

Table 1
OpenSSL energy efficiency comparison of EECA, Software, instruction set and hardware accelerator.

Block size (kB)	EECA		Software			Instruction set			Hardware accelerator		
	TP (kB/s)	Power (W)	TP (kB/s)	Power (W)	EECA PPW imprv.	TP (kB/s)	Power (W)	EECA PPW imprv.	TP (kB/s)	Power (W)	EECA PPW imprv.
1	5 506 942	139.622	647 447	142.295	766.84%	5 912 479	140.779	−6.09%	556 045	141.072	966.23%
2	5 765 027	139.588	650 029	142.333	804.32%	5 986 630	140.819	−2.85%	1 110 012	141.184	479.47%
4	5 862 733	139.459	651 133	142.174	817.92%	6 025 041	140.883	−1.70%	2 175 144	141.526	204.51%
8	5 946 091	139.539	650 857	142.380	832.18%	6 037 716	141.109	−0.41%	4 110 016	141.717	62.72%
16	7 113 426	140.391	649 313	142.208	1009.71%	6 043 481	140.762	18.02%	6 095 124	141.003	19.77%
32	8 996 049	140.953	647 004	142.527	1305.94%	6 047 764	141.115	48.92%	6 066 890	139.571	46.63%
64	11 010 608	141.770	646 754	142.577	1612.13%	6 049 740	141.188	81.25%	6 055 810	138.783	76.80%
128	11 112 462	141.507	646 624	142.555	1631.26%	6 051 503	141.358	83.44%	6 032 583	138.342	77.89%
256	11 155 597	141.811	645 110	142.457	1637.13%	6 033 402	141.750	84.82%	6 271 619	138.050	77.67%

size checkout and scheduling decision. Therefore, for small data block, the maximum performance of *EECA* is somewhat lower than instruction set implementation. However, this influence is decreased with the increase of data size, on account of the reduced frequency for scheduling checkout.

(3) ***EECA* vs. hardware encryption**
As shown in Fig. 9 the energy efficiency of hardware encryption is worse for small data block. Smaller the data block, worse the energy efficiency. As we can see from the results in Table 1, the throughput of *EECA* remarkably outperforms hardware accelerator method. For small data block that less than 8 kB, *EECA* could get several times PPW improvement comparing with hardware accelerator encryption. Smaller the data block, bigger the improvement. The reason behind this is the adaptive scheduling avoiding low performance of accelerators. For large data blocks, which typically performs well for hardware accelerators, the proposed *EECA* still could get around 19.77% to 77.89% PPW over hardware accelerator implementation. This contribution comes from best utilization of system resource through *Dynamic Management Mechanism*.

5.4. Web server energy efficiency

We adopt pressure test *ab* to evaluate the energy efficiency of the whole Web server for serving the HTTPS requests with cipher suite ECDHE-RSA-AES256-SHA384 when using software, instruction set, hardware accelerator and *EECA*, respectively. We record the RPS and energy consumption (Power) of different encryption methods in the condition of variable page size, including 1 kB, 2 kB, 8 kB, 16 kB, 32 kB, 64 kB, 128 kB, 256 kB, 1 MB and 2 MB. For a clear comparison and analysis, we transit RPS to network bandwidth (kB/s) and divide it by corresponding energy consumption to get the PPW as shown in Fig. 10. The PPW of *EECA* increased from 418.37 kB/W to 14 189.82 kB/W. Regardless of page size, *EECA* makes the utmost of CPU and hardware accelerators resource. To better present the HTTPS energy-efficiency improvement with our proposed *EECA*, we show the incremental multiples of PPW for *EECA* (Imprv.) comparing with software, instruction set and hardware in Table 2.

(1) ***EECA* vs. software implementation**
As we discussed before, software encryption has a poor performance for no acceleration. Thus, for the whole Web server system, it can only have a general energy efficiency. The PPW improvement of *EECA* against the original Web server with software encryption is 21.34% to 414.37%.

(2) ***EECA* vs. instruction set implementation**
When data size is larger than 32 kB, the PPW improvement of *EECA* against the original Web server with instruction set is 1.25% to 40.14%. Beyond that, the PPW with *EECA* is a little bit lower than instruction set. As we discussed before,

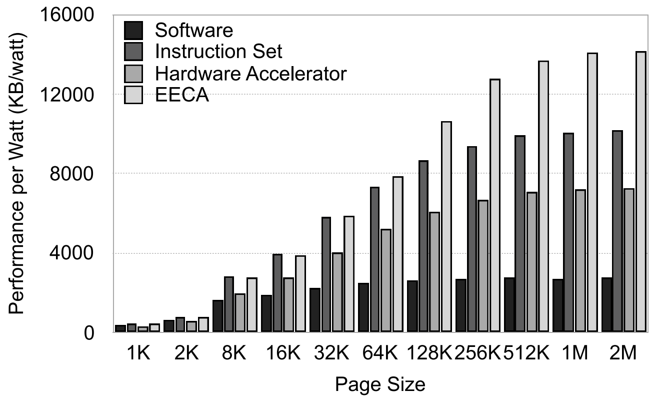


Fig. 10. Energy efficiency of 4 different encryption methods with cipher suite ECDHE-RSA-AES256-SHA.

the cost for request scheduling detection and management has a slight impact on performance. Therefore, for the small page size, the performance of *EECA* is somewhat lower than instruction set implementation. However, this influence is decreased with the increase of page size, on account of the reduced frequency for scheduling checkout.

(3) ***EECA* vs. hardware accelerators implementation**
The PPW improvement of *EECA* against the original Web server with hardware accelerator is 93.22% to 96.05%. As we can see from Fig. 10, *EECA* can get greater PPW compared with original Web server with hardware encryption. The reason is due to great reduction of invocation cost through proposed *Request Reconstruction* and *Dynamic Management Mechanism*. The influence is more obvious for large page sizes. For example, the maximum improvement achieves 96.05% for case page 1 MB.

5.5. Experimental conclusion

Based on the above experiments and analysis, we can get the following conclusions. *EECA* can significantly improve energy efficiency for Web server, regardless of the request data size. On one hand, *Dynamic Management Mechanism* cooperating hardware accelerators and instruction set in *EECA* exerts their respective superiorities and makes full use of system resource which contributes a better energy efficiency for encryption in the case of various data size. On the other hand, *Request Reconstruction* can further improve the energy efficiency for Web server. When a client requests for a large data, the whole plaintext of the request invoke the hardware accelerator one time to reduce the overhead of hardware invocation.

Table 2
HTTPS energy efficiency comparison of EECA, software, instruction set and hardware accelerator.

Block size	EECA		Software			Instruction set			Hardware accelerator		
	TP (kB/s)	Power (W)	TP (kB/s)	Power (W)	EECA PPW imprv.	TP (kB/s)	Power (W)	EECA PPW imprv.	TP (kB/s)	Power (W)	EECA PPW imprv.
1 kB	59 241	141.600	48 877	141.765	21.34%	61 047	141.464	−3.05%	41 373	141.522	43.11%
2 kB	54 849	141.580	42 441	141.789	29.42%	56 001	141.463	−2.14%	38 596	141.554	42.09%
8 kB	48 705	141.698	28 519	141.842	70.95%	49 956	141.489	−2.65%	34 939	141.701	39.40%
16 kB	34 158	141.823	16 751	142.032	104.21%	34 718	141.573	−1.79%	24 530	141.791	39.22%
32 kB	26 040	142.175	9 992	142.163	160.57%	25 609	141.568	1.25%	17 876	141.882	45.37%
64 kB	17 468	142.609	5 525	142.208	215.27%	16 240	141.596	6.80%	11 476	141.981	51.54%
128 kB	11 881	143.173	2 916	142.334	305.05%	9 542	141.594	23.14%	6 764	142.066	74.31%
256 kB	7 175	144.121	1 493	142.362	374.62%	5 204	141.656	35.52%	3 695	142.127	91.50%
512 kB	3 863	144.308	761	142.392	400.71%	2 737	141.635	38.52%	1 953	142.126	94.79%
1 MB	1 984	144.417	375	142.415	422.26%	1 389	141.640	40.14%	997	142.179	96.05%
2 MB	1 001	144.417	192	142.433	414.37%	705	141.593	39.18%	505	142.046	94.98%

6. Conclusion

In this paper, we reveal the serious energy-efficient problem of HTTPS with state-of-the-art encryption approaches. We find that instruction set and hardware accelerator cannot be efficiently utilized by Web server to improve the PPW. One of the reasons is that the encryptions are all handled using small data size (no more than 16 kB), which poses considerable management overhead to call hardware. The other one is that the limited CPU resource becomes the bottleneck for further PPW improvement for instruction set. In this paper, we propose a novel scheme to improve the energy efficiency of HTTPS secure access with OpenSSL through HW-SW co-design, called *EECA*. The essential idea is to use respective energy-efficient advantages of instruction set and hardware accelerator for different cases. In order to take full advantage of system resource, we propose to reorganize and schedule HTTPS requests according to the data size. The proposed *Request Reconstruction* reducing the overhead of calling hardware by submitting request data for encryption at once. We design the *Adaptive Control Crypto* (*ac_crypto*) engine with a *Dynamic Management Mechanism*, including *RequestAllocation* algorithm and *DynamicScheduler* algorithm. It exerts the superiorities of the instruction set and hardware and makes full use of system resource. The *RequestAllocation* algorithm divides the encryption requests into different task queues according to the data size of each request, meanwhile the *DynamicScheduler* algorithm dynamically manages these requests between these task queues. Evaluations show that the proposed *EECA* can improve the energy efficiency of Web server by up to 422.26%, 40.14% and 96.05% comparing with the original Web server using software, instruction set and hardware accelerators, respectively. The experimental results also show that, for typical encryption algorithm AES-256-CBC, the proposed *EECA* can provide the energy efficiency of encryption improvement by up to 1637.13%, 84.82%, and 966.23% comparing with software encryption, instruction set and hardware encryption, respectively. Proposed design methodology possesses universal properties. It could be applicable to other applications with different kinds of encryption algorithm but differ in effect.

References

[1] D. Kline, N. Parshook, X. Ge, E. Brunvand, R. Melhem, P.K. Chrysanthis, A.K. Jones, Holistically evaluating the environmental impacts in modern computing systems, in: Green and Sustainable Computing Conference, 2017, pp. 1–8.
[2] S. Subashini, V. Kavitha, A survey on security issues in service delivery models of cloud computing, J. Network Comput. Appl. 34 (1) (2011) 1–11.
[3] K. Gai, K.-K.R. Choo, M. Qiu, L. Zhu, Privacy-preserving content-oriented wireless communication in internet-of-things, IEEE Internet Things J. (2018).
[4] K. Gai, M. Qiu, Blend arithmetic operations on tensor-based fully homomorphic encryption over real numbers, IEEE Trans. Ind. Inf. 14 (8) (2018) 3590–3598.
[5] Z. Jia, X. Tian, A novel security private cloud solution based on ecryptfs, in: Information Management, Innovation Management and Industrial Engineering (ICIII), 2013 6th International Conference on, Vol. 3, IEEE, 2013, pp. 38–41.
[6] J. Wilson, R.S. Wahby, H. Corrigan-Gibbs, D. Boneh, P. Levis, K. Winstein, Trust but verify: auditing the secure internet of things, in: Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services, ACM, 2017, pp. 464–474.
[7] J. Viega, M. Messier, P. Chandra, Network Security with OpenSSL: Cryptography for Secure Communications, " O'Reilly Media, Inc.", 2002.
[8] E.A. Young, T.J. Hudson, R. Engelschall, Openssl: The open source toolkit for ssl/tls, 2011.
[9] S. Gueron, Intel Advanced Encryption Standard (AES) Instruction Set White Paper., September 2012.
[10] I. McCallum, Intel QuickAssist Technology Accelerator Abstraction Layer (AAL), Intel Corporation, 2007.
[11] H. Klemick, E. Kopits, A. Wolverton, et al., Data center energy efficiency investments: qualitative evidence from focus groups and interviews, in: Tech. rep., National Center for Environmental Economics, US Environmental Protection Agency, 2017.
[12] J. Lenhardt, W. Schiffmann, Energy efficient processing of fine-grained loads in heterogeneous server farms, in: Computing and Networking (CANDAR), 2016 Fourth International Symposium on, IEEE, 2016, pp. 105–111.
[13] C.G.C. Index, Forecast and methodology, 2015-2020 white paper, Retrieved 1st June, 2016.
[14] A. Shehabi, S. Smith, D. Sartor, R. Brown, M. Herrlin, J. Koomey, E. Masanet, N. Horner, I. Azevedo, W. Lintner, United states data center energy usage report, 2016.
[15] C. Reams, Modelling Energy Efficiency for Computation (Ph.D. thesis), University of Cambridge, 2012.
[16] A. Hooper, Green computing, Commun. ACM 51 (10) (2008) 11–13.
[17] S. Murugesan, Harnessing green it: principles and practices, IT Professional 10 (1) (2008).
[18] A. Freire, C. Macdonald, N. Tonellotto, I. Ounis, F. Cacheda, A self-adapting latency/power tradeoff model for replicated search engines, in: Proceedings of the 7th ACM international conference on Web search and data mining, ACM, 2014, pp. 13–22.
[19] N.R. Potlapally, S. Ravi, A. Raghunathan, N.K. Jha, A study of the energy consumption characteristics of cryptographic algorithms and security protocols, IEEE Trans. Mob. Comput. 5 (2) (2006) 128–143.
[20] A.K. Jones, Green computing: new challenges and opportunities, in: Proceedings of the on Great Lakes Symposium on VLSI 2017, ACM, 2017, 3–3.
[21] R. Evans, J. Gao, DeepMind AI Reduces Google Data Centre Cooling Bill by 40%, 2016, URL https://deepmind.com/blog/deepmind-ai-reduces-google-data-centre-cooling-bill-40/.
[22] L.A. Barroso, The price of performance, Queue 3 (7) (2005) 48–53.
[23] J. Laudon, Performance/watt: the new server focus, ACM SIGARCH Comput. Archit. News 33 (4) (2005) 5–13.
[24] E. Rescorla, Http over tls, 2000.
[25] A. Hafsa, N. Alimi, A. Sghaier, M. Zeghid, M. Machhout, A hardware-software co-designed aes-ecc cryptosystem, in: Advanced Systems and Electric Technologies (IC_ASET), 2017 International Conference on, IEEE, 2017, pp. 50–54.
[26] L. Batina, D. Hwang, A. Hodjat, B. Preneel, I. Verbauwhede, Hardware/software co-design for hyperelliptic curve cryptography (hecc) on the 8051 μp, in: International Workshop on Cryptographic Hardware and Embedded Systems, Springer, 2005, pp. 106–118.
[27] A. Nadjia, A. Mohamed, Aes ip for hybrid cryptosystem rsa-aes, in: Systems, Signals & Devices (SSD), 2015 12th International Multi-Conference on, IEEE, 2015, pp. 1–6.
[28] M.A. Hasamnis, S. Limaye, Design and implementation of rijindael's encryption algorithm with hardware/software co-design using nios ii processor, in: Industrial Electronics and Applications (ICIEA), 2012 7th IEEE Conference on, IEEE, 2012, pp. 1386–1389.
[29] S. Baskaran, P. Rajalakshmi, Hardware-software co-design of aes on fpga, in: Proceedings of the International Conference on Advances in Computing, Communications and Informatics, ACM, 2012, pp. 1118–1122.
[30] M.A. Hasamnis, S. Limaye, An approach to design advanced standard encryption algorithm using hardware/ software co-design methodology, Int. J. Eng. Sci. Technol. 4 (5) (2012).

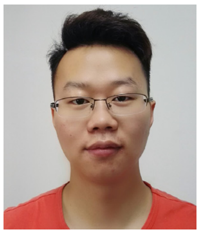
[31] L. Amaral, G. Araujo, J. López, Hw/sw co-design of identity-based encryption using a custom instruction set, in: Field-Programmable Technology, 2009. FPT 2009. International Conference on, IEEE, 2009, pp. 510–513.
[32] C.-H. Lien, Y.-W. Bai, M.-B. Lin, P.-A. Chen, The saving of energy in web server clusters by utilizing dynamic sever management, in: Networks, 2004.(ICON 2004). Proceedings. 12th IEEE International Conference on, Vol. 1, IEEE, 2004, pp. 253–257.
[33] T. Horvath, K. Skadron, T. Abdelzaher, Enhancing energy efficiency in multi-tier web server clusters via prioritization, in: Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International, IEEE, 2007, pp. 1–6.
[34] E.M. Elnozahy, M. Kistler, R. Rajamony, Energy-efficient server clusters, in: International Workshop on Power-Aware Computer Systems, Springer, 2002, pp. 179–197.
[35] A. Beloglazov, R. Buyya, Y.C. Lee, A. Zomaya, A taxonomy and survey of energy-efficient data centers and cloud computing systems, in: Advances in Computers, Vol. 82, Elsevier, 2011, pp. 47–111.
[36] J. Lenhardt, K. Chen, W. Schiffmann, Energy-efficient web server load balancing, IEEE Syst. J. 11 (2) (2017) 878–888.
[37] Y. Yang, N. Xiong, A. Aikebaier, T. Enokido, M. Takizawa, Minimizing power consumption with performance efficiency constraint in web server clusters, in: Network-Based Information Systems, 2009. NBIS'09. International Conference on, IEEE, 2009, pp. 45–51.
[38] T. Horvath, T. Abdelzaher, K. Skadron, X. Liu, Dynamic voltage scaling in multitier web servers with end-to-end delay control, IEEE Trans. Comput. 56 (4) (2007) 444–458.
[39] K. Gai, M. Qiu, Z. Ming, H. Zhao, L. Qiu, Spoofing-jamming attack strategy using optimal power distributions in wireless smart grid networks, IEEE Trans. Smart Grid 8 (5) (2017) 2431–2439.
[40] L. Gui-hong, Z. Hua, L. Gui-zhi, Building a secure web server based on openssl and apache, in: E-Business and E-Government (ICEE), 2010 International Conference on, IEEE, 2010, pp. 1307–1310.
[41] L. Bossuet, M. Grand, L. Gaspar, V. Fischer, G. Gogniat, Architectures of flexible symmetric key crypto enginesa survey: from hardware coprocessor to multi-crypto-processor system on chip, ACM Comput. Surv. 45 (4) (2013) 41.
[42] C.B. Barth, I. Moon, Y. Lei, S. Qin, C. Robert, et al., Experimental evaluation of capacitors for power buffering in single-phase power converters, in: Energy Conversion Congress and Exposition (ECCE), 2015 IEEE, IEEE, 2015, pp. 6269–6276.
[43] Y.D.P. Meter, WT310/WT310HC/WT330 Digital Power Meter User's Manual, 2013.
[44] Apache, ab - Apache HTTP server benchmarking tool., October 2014.
[45] A.J. Stieber, OpenSSL hacks, Linux J. (147) (2006) 74–77.

Chunhua Xiao, born in 1987. She is currently an associate professor at School of Computer Science, Chongqing University, China. She received the Ph.D. degree from the Beijing University of Technology, China. She did one year (2011–2012) academic research as a joint-training Ph.D. student in University of California, Los Angeles. Dr. Xiao has authored and co-authored more than 30 publications in peer-reviewed journals and conferences. She has been an independent PI for a standard (2016–2018) NSFC(National Nature Science Foundation of China) grant, and also an independent PI (2016–2017) for a Crossing Research Projects with Huawei Technologies Co. Ltd; She was honored with Science and Technology Progress Award from Beijing municipality in the year 2012. Her research interests include MPSoCs, hardware and software co-design, and embedded systems.

Lei Zhang is currently pursuing the master's degree under the supervision of Dr. C. Xiao with Chongqing University. She received the B.E. degree from the School of Computer Science and Technology, Chongqing University of Posts and Telecommunications, Chongqing, China, in 2016. Her current research interests include hardware security, hardware and software co-design, and energy-efficient computing and applications.

Weichen Liu (S07-M11) is an assistant professor at School of Computer Science and Engineering, Nanyang Technological University, Singapore. He received the Ph.D. degree from the Hong Kong University of Science and Technology, Hong Kong, and the BEng and MEng degrees from Harbin Institute of Technology, China. Dr. Liu has authored and co-authored more than 70 publications in peer-reviewed journals, conferences and books, and received the best paper candidate awards from ASP-DAC 2016, CASES 2015, CODES+ISSS 2009, the best poster awards from RTCSA 2017, AMD-TFE 2010, and the most popular poster award from ASP-DAC 2017. His research interests include embedded and real-time systems, multiprocessor systems and network-on-chip.

Neil Bergmann has been Professor of Embedded Systems in the School of ITEE at The University of Queensland since 2001. He has Bachelor degrees in Engineering, Science and Arts from University of Queensland, and a Ph.D. in Computer Science from the University of Edinburgh, UK (1984). He is a member of IEEE and a Fellow of the Institution of Engineers, Australia. His research interests are in computer systems, especially reconfigurable computing and wireless sensor networks.



Yuhua Xie is currently pursuing the master's degree under the supervision of Dr. C. Xiao with Chongqing University. He received the B.E. degree from the School of Computer Science and Technology, Chongqing University of Posts and Telecommunications, Chongqing, China, in 2015. His research interests include NoCs based on emerging interconnect technology and information security.