Article

Client-Aware Negotiation for Secure and Efficient Data Transmission

Ziheng Wang, Heng Chen * and Weiguo Wu

School of Computer Science and Technology, Xi'an Jiaotong University, Xi'an 710049, China;
wzh009888@stu.xjtu.edu.cn (Z.W.); wgwu@xjtu.edu.cn (W.W.)

* Correspondence: hengchen@xjtu.edu.cn

Received: 12 October 2020; Accepted: 2 November 2020; Published: 4 November 2020

Abstract: In Wireless Sensor Networks (WSNs), server clusters, and other systems requiring secure transmission, the overhead of data encryption and transmission is often not negligible. Unfortunately, a conflict exists between security and efficiency in processing data. Therefore, this paper proposes a strategy to overcome this conflict, called Client-Aware Negotiation for Secure and Efficient Data Transmission (CAN-SEAT). This strategy allows a client with different security transmission requirements to use the appropriate data security transmission without modifying the client. Two methods are designed for different clients. The first method is based on two-way authentication and renegotiation. After handshakes, the appropriate data security transmission scheme is selected according to the client requirements. Another method is based on redirection, which can be applied when the client does not support two-way authentication or renegotiation. For the characteristics of different architecture, this paper classifies and discusses symmetric key algorithms, asymmetric key algorithms, and hardware encryption instructions. In four application scenarios, the CAN-SEAT strategy is tested. Compared with the general transmission strategy, when only software encryption is used, the data processing and transmission cost can be reduced by 89.41% in the best case and by 15.40% in the worst case. When supporting hardware encryption, the cost can be reduced by 85.30% and 24.63%, respectively. A good effect was produced on the experimental platforms XiLinux, FT-2000+, and Intel processors. To the best of our knowledge, for Client-Aware Negotiation (CAN), this is the first method to be successfully deployed on a general system. CAN-SEAT can be easily combined with other energy-efficient strategies.

Keywords: SSL/TLS protocol; data security transmission; two-way authentication; client-aware negotiation; energy saving

1. Introduction

As a security layer protocol, the Secure Socket Layer (SSL) and the subsequent Transport Layer Security (TLS) are widely used in TCP/IP networks. SSL/TLS is often combined with the Hypertext Transfer Protocol (HTTP) to ensure the security of data transmission; in this case, HTTP is also known as Hypertext Transfer Protocol Secure (HTTPS). Compared to the traditional transmission mode, data transmission based on SSL/TLS needs to introduce additional security control overhead. The efficiency and security are closely related. Usually, more efficiency means less security; conversely, more security means less efficiency and, more broadly, higher energy consumption. In practical applications, different control strategies can be set for data transmission with different security requirements.

1.1. Background and Motivation

In Wireless Sensor Networks (WSNs), encryption is usually used to ensure the integrity and security of data transmission. We can consider an application scenario where multiple levels of secure data transmission strategies are set for different sensors to reduce the cost of security control, thus reducing node energy consumption and ultimately prolonging the lifetime of the WSN. Meanwhile, the server in server clusters can set the different security strength for different transmissions to reduce the resource consumption of the server. When the load of the server cluster is reduced, power consumption can be reduced by shutting down the machine.

In WSNs, many studies are related to energy saving transmission. We can combine these approaches to apply the Client-Aware Negotiation for Secure and Efficient Data Transmission (CAN-SEAT) strategy. Mehmood et al. [1] designed an energy-efficient routing protocol to extend the lifetime of a WSN called LEACH-VH, which can save energy using selective sleeping nodes and backup nodes. Kuila et al. [2] focused on two main optimization problems in WSNs: energy-efficient clustering and routing. They used an efficient particle encoding scheme and a multi-objective fitness function to solve the routing and the clustering problems by load balancing the energy conservation of the nodes. Wei et al. [3] considered that the hybrid Software-Defined Wireless Rechargeable Sensor Networks (SDWRSNs) will last for a long time and introduced an Energy-saving Traffic Scheduling (ETS) algorithm to improve the energy efficiency of this network. Liu et al. [4] were motivated by the requirement of heterogeneity in the Internet of Things and discovered several regularities. Ebrahimi et al. [5] studied data gathering problems. Specifically, they optimized the joint application of compressive sensing and network coding by optimally constructing forwarding trees to carry compressed data to projection nodes. Afterwards, Yu et al. [6] used a sparse random matrix as the sensing matrix to solve the issue of compressive sensing. At the same time, they greedily constructed a sequence of a data gathering tree to collect linearly combined measurements, and their method was proven to be near-optimal. Qiu et al. [7] creatively presented a design principle called talk more listen less to reduce idle-listening in neighbor discovery protocols. They designed an energy-efficient asynchronous neighbor discovery protocol named Nihao for both symmetric and asymmetric cases.

The CAN-SEAT strategy is also applicable to the server cluster, and it can be combined with the server's energy-saving approach described below. Yin et al. [8] proposed a strategy that switches servers between on and sleep in groups based on multi-server vacation queuing theory. Kuehn et al. [9] provided a model to manage the energy consumption of the server via load-dependent server activation and sleep modes. They defined a multi-server queuing model controlled by a finite state machine to analyze the energy efficiency and performance. Numerical result were provided, and the applicability of the model was proven. Nguyen et al. [10] studied the queuing system of a data center. They provided two methods for active thresholds to minimize power consumption depending on the current queue size relative to the threshold, and the data center switches specific groups of servers to on or off.

To implement client-aware secure transmission, the ideal approach is to achieve customization, i.e., to modify both the client and the server. However, in the actual deployment process, the client is often difficult or impossible to modify. For example, when the client accesses the server through the browser, due to the variety and version of the browser and as the browser has become general software, the CAN-SEAT strategy can only be deployed on the server in this case. However, servers using the HTTPS protocol currently do not provide support for client-aware secure transmission. Therefore, for the system where only part of the data has higher security requirements, there will be more computing overhead and energy consumption. To solve this problem, this paper proposes two client-aware negotiation methods: one based on renegotiation and the other based on redirection. The CAN-SEAT strategy aims at reducing the cost of the existing secure transmission protocols. It can be incorporated into many practical systems. The CAN-SEAT strategy is only described as an example built on the HTTPS protocol and can be deployed with other protocols depending on the specific implementation of the protocols.

1.2. Related Work

The CAN-SEAT strategy is based on an authentication mechanism. In different application scenarios, the authentication mechanism is not exactly the same. For most systems that require authentication, there are some authentication mechanisms based on Public Key Cryptography (PKC), which is called a full handshake in SSL/TLS. At the same time, in SSL/TLS, there will be some authentication mechanisms for quick recovery. The usual purpose of these mechanisms is to reduce the overhead and response time for authentication on the basis of ensuring security. The traditional approach is session resume [11], and information from both communicating parties is stored locally for a period of time to speed up authentication. In addition, some new technologies are used in the new version to reduce overhead, such as 0-RTT and pre-shared key [12].

Security and privacy issues are also some of the major challenges in the Internet of Things (IoT). Authentication, as an important part of ensuring security, uses the certificate to confirm the identity in the IoT. The traditional authentication methods are enhanced with authentication based on the user's location. Gergely et al. [13] suggested base authentication on attributes, rather than identities, to enhance privacy. Kumar et al. [14] considered that the biometric information of the owner should be used for authentication. Pinto et al. [15] proposed a mechanism of HTTP(S)-based authentication for IoT devices using a hash-chain generated between the server and the client. However, no matter how the identity is verified, the authentication process is based on PKC.

There are many energy-efficient strategies for secure data transmission. Gai et al. [16] investigated the problem where the current implementations of intelligent transportation systems are restricted by the conflicts between security and communication efficiency. Therefore, they proposed a security-aware efficient data sharing and transferring model to obtain secure real-time multimedia data sharing and transfer. Shang et al. [17] designed a special secure transmission scheme called Energy-Efficient Transmission (EET) for the Internet of Things by considering the direct link that can be adopted in practical applications as an additional strategy. Zhang et al. [18] studied a Secrecy Energy Efficient (SEE) optimization problem in a multiple-input, multiple-output non-orthogonal multiple access network system. They proposed several iterative algorithms to solve the original SEE maximization problem based on Dinkelbach's algorithm and the difference of concave approximation method. To solve the same problem, Wang et al. [19] proposed a fractional programming approach embedded with the difference of convex functions programming to manage the non-convexity of the formulated problem. The core idea is that the primal problem can be translated into a series of convex subproblems and then be solved by convex programming.

Although all of the above work suggests some energy-efficient strategies for secure data transmission, their methods are more dedicated and difficult to apply to the server or HTTPS. Therefore, research on HTTPS is still necessary. In related work, the authors did not find a system that uses the method described in this manuscript. More related research is based on algorithms and protocols. However, distinguishing users is indeed an effective and feasible solution for reducing transmission overhead.

2. System Description

The CAN-SEAT strategy is based on HTTPS, which consists of HTTP and SSL/TLS. Its main purpose is to provide authentication to the server and ensure the security and integrity of data transmission. Existing mainstream HTTP servers include Apache, Nginx, IIS, and so on. As an open-source project, the Apache server has always maintained a high market share [20]. The SSL/TLS implementation has OpenSSL, GunSSL, wolfSSL, and so on. As an open-source implementation of SSL/TLS, OpenSSL has become the de facto standard for secure data transmission. The SSL/TLS implementation usually is comprised of two main parts: the SSL protocol library and the cryptographic algorithm library. The SSL protocol library mainly includes various SSL/TLS protocol implementations, and the cryptographic algorithm library contains many kinds of algorithms for data encryption.

2.1. Message Exchange

The SSL protocol library contains a variety of protocols. Among them, the handshake protocol is necessary to ensure the security of data transmission, being used to negotiate secure transmission parameters before secure data transmission. There are three common handshakes: (1) a full handshake, named one-way authentication, where only the server is authenticated; (2) a full handshake, named two-way authentication, where both the client and the server are authenticated; and (3) a short handshake, named session resume, which is used to restore the previous session with a small cost.

Figure 1 shows the message exchange of a full handshake; it can be used to describe both one- and two-way authentication. They both require server authentication. The solid line indicates the process of one-way authentication; the dotted line describes the message passing that is unique to two-way authentication, which needs to authenticate the client. The messages with * represent messages that are used only during two-way authentication. Both the client and server are authenticated by means of a certificate; the failure of authentication directly leads to the failure of the handshake, which will then terminate the connection. The specific meanings and the latest details of the protocol can be found in [12].

Figure 1. A flow diagram of client-aware secure data transmission methods. The messages with '*' are for two-way authentication.

Figure 1 shows the processing phase of the CAN-SEAT strategy. After the first handshake, according to the identity of the client, the corresponding secure transmission scheme is deployed. The strategy aims at the data transmission encryption algorithm, mainly at the asymmetric encryption and symmetric encryption algorithm. In addition to the method shown here, this article provides an alternative method based on redirection to support different clients. Finally, as shown in Figure 1, after the completion of the deployment of the CAN-SEAT strategy, the secure data transmission starts.

2.2. Encryption Algorithm

In the process of handshake and secure data transmission, different encryption algorithms are used in different phases. For HTTPS, encryption algorithms are required for key exchange, authentication, data encryption, and signature, but the encryption algorithms applied to each part are different. The strength of the algorithm is closely related to the efficiency of encryption. The CAN-SEAT strategy differentiates clients by differentiating these encryption algorithms.

Key exchange: In the SSL/TLS protocol, the key exchange is used to generate the key for data transmission encryption. Because of the higher security requirements, key exchange requires the use of asymmetric encryption algorithms. The algorithms actually used are RSA, Diffie–Hellman Ephemeral (DHE), and Elliptic Curve Diffie–Hellman Ephemeral (ECDHE).

Authentication: At the start of the session, the SSL/TLS protocol may require both communicating ends to prove their identity, typically using digital signatures for authentication. Asymmetric encryption is also required for authentication. The actual algorithm includes RSA and ECDSA. In practice, to avoid the cryptographic operations being repeatedly performed, authentication and key exchange are generally used in combination before data transmission.

Data encryption: After establishing the connection, it is necessary to encrypt the data transmission. Because the amount of data is usually large, the data are encrypted using an efficient symmetric encryption algorithm, including the Triple Data Encryption Algorithm (3DES), the Advanced Encryption Standard (AES), CAMELLIA, ARIA, and ChaCha20, which remain relatively secure. There are three encryption methods: stream encryption, block encryption, and Authentication Encryption with Associated Data (AEAD). At present, ChaCha20 is the only secure stream encryption algorithm currently in use for SSL/TLS. Regarding block encryption, because of the low encryption efficiency of 3DES, the actual algorithm used are AES, CAMELLIA, ARIA, etc. Among them, AES is still the most common algorithm. Compared with the same algorithm in the previous two methods, AEAD is more efficient and secure, but it is only supported in the new version (TLSv1.2 and subsequent versions). Currently, in TLSv1.3, the algorithms only include ChaCha20 and AES [21].

Signature: In SSL/TLS, the common hash signature algorithms include the Message-Digest Algorithm (MD5), Secure Hash Algorithm 1 (SHA1), SHA, SHA256, and SHA384 with the Hash-based Message Authentication Code (HMAC). In previous versions of TLSv1.2, the signature algorithm was applied to the Message Authentication Code (MAC) and the Pseudorandom Function (PRF) algorithm. In later versions, the PRF algorithm must be used.

2.3. Security Strength

The security strength of the encryption algorithm is related to the encryption overhead, so knowing the strength of the encryption algorithm can help to better select the appropriate encryption algorithms in the actual system. Table 1 illustrates the minimal security key sizes for different algorithms. The common symmetric encryption algorithms are AES, CAMELLIA, ARIA, etc. The common factoring modules algorithms include RSA, and the common algorithms for elliptic curves include ECDSA. Hash functions are implemented in TLS with SHA, for example. The key size of the algorithm is mostly reflected in the name of the algorithm, and some key sizes of the algorithms that are not clearly reflected can be found in [22].

Table 1. The minimal key sizes for security (bits) [23].

Protection	Symmetric	Factoring Modulus	Elliptic Curve	Hash
Legacy standard level Should not be used in new systems	80	1024	160	160
Near-term protection Security for at least 10 years (2020–2028)	128	3072	256	256
Long-term protection Security for 30 to 50 years (2020–2068)	256	15360	512	512

3. Methodology

Using the CAN-SEAT strategy, clients may need to provide a valid digital certificate to identify themselves. The client is represented as $C = \{n, e, s, v\}$, where n , e , s , and v represent the client's name,

e-mail, issuer information, and the certificate validity data, respectively. This section describes two methods of client-aware negotiation based on renegotiation or redirection.

3.1. CAN Based on Renegotiation

Typically, in the SSL/TLS protocol, the data transmission can begin once the handshake completes, but due to the limitation of the handshake protocol, it may take two handshakes before the data can be transmitted. The specific reason is shown in Figure 1. Negotiation of the encryption algorithm is completed in the server hello message. However, the client's certificate is sent to the server in the certificate message. Therefore, the identity information of the client cannot be obtained before negotiation.

The flowchart of CAN based on renegotiation is shown in Figure 2. Firstly, the methods need to determine whether the handshake is completed, which means, in the near future, whether the session still needs to negotiate the way of communication. If the handshake is completed, the data transmission begins immediately after the session resume is completed. Session resume is used for the low-cost and fast resume of sessions, and data transmission can be carried out directly. If the handshake is not completed, the method needs to determine how many handshakes have been completed. If the current handshake is the first handshake, the client's certificate needs to be obtained, and the client's identity information needs to be obtained from the certificate. If the client being accessed needs the highest security strength of data transmission, renegotiation is not needed because the encryption security strength of the first handshake in the strategy is the highest. After completing the above operations by the client needing the highest security strength, the strategy considers the handshake as finished, and the data transmission starts.

Figure 2. A flowchart of client-aware negotiation based on renegotiation.

If a client does not need the highest level of security, first, the number of handshakes is changed in the corresponding position of the handshake information table. After that, the client information table is searched, then the corresponding encryption scheme is found and deployed. Finally, the server initiates the renegotiation, asking the client to re-initiate the handshake. When the client allows the renegotiation, the client initiates a second handshake. After a series of message exchanges, the client and server complete a second two-way authentication. Then, if the certificates of both parties meet the requirements, data transmission begins.

The server cannot obtain the client's certificate before the handshake, so it cannot obtain the client's identity information. At this point, a unique connection can be determined by the IP and port of the client. During the server cache time, which means some client's information such as the number of handshakes is still stored in the server, a client corresponds to only one connection, and a connection serves only one client. Thus, clients can be differentiated by a connection, which makes it possible to determine how many handshakes the connection has completed with the second handshake. The connection information is stored in the handshake information table.

Both the handshake information table and the client information table have their own processing mechanisms. The handshake information table maintains clients' handshake information, which is used to record the handshake number of the connection and whether the handshake was successful. This table needs to periodically remove redundant data; the storage time of the handshake information is the same as the server cache time. When the system deletes some cached data, the clients' handshake information is also deleted.

The client information table records the client's corresponding secure data transmission scheme. Unlike the handshake information table, the client information table is stored as a file in the shared area. If the administrators want to change user information, just the file needs to be changed. The user information is not stored on the server.

Take the actual scenario of deploying OpenSSL on the Apache server as an example. To handle HTTPS transmission, the Apache server interacts with OpenSSL through the default module of Apache: mod_ssl. In OpenSSL, an SSL structure is used to represent an SSL connection. The structure preserves the information during a handshake and cleans up some messages after a handshake is complete. After a certain amount of time, the connection information is cleared by the server.

The implementation of the SSL/TLS handshake protocol is controlled by OpenSSL, specifically by the function `SSL_Accept` in OpenSSL. This function is responsible for the overall processing of the handshake protocol and realizes the handshake message exchange through the state machine mechanism. One-way authentication, two-way authentication, and session resume are all realized by function `SSL_Accept` through the state machine mechanism.

In theory, the CAN method based on renegotiation can be implemented by modifying `SSL_Accept` in OpenSSL. However, because the implementation of OpenSSL is not inherited and varies widely from version to version, it is less compatible due to the large differences between versions. For example, there are considerable differences between OpenSSL 1.0.2, OpenSSL 1.1.0, and OpenSSL 1.1.1. The implementation at the Apache server level is inherited, so this method can be applied to different versions of OpenSSL. The scenario discussed in this section was deployed on the Apache server and revolves around the handshake.

The Apache server controls the handshake through the Input Output (IO) module in mod_ssl, which is used at the beginning of both receiving and sending data. For the server, the handshake is the first behavior initiated by the client, so the handshake function is executed first in the input function. In the first handshake, the client certificate is sent to the server via a certificate message. For standard client certificates in X509 format, Apache stores certificate information in the SSL structure. In the client's certificate, the `subject_name` field contains the client identity information $C = \{n, e, s, v\}$. Therefore, this field can be used to obtain the identity information of the client and the corresponding security transmission policy of the client through the client information table. These secure transmission schemes are embodied in OpenSSL as cipher suites and certificates for

clients to set up key exchange, authentication, data encryption, and signature. The CAN-SEAT strategy ensures that the corresponding secure transmission scheme is used for the next SSL handshake and data transmission by setting the corresponding cipher suite for the SSL connection and encrypting the certificate using the asymmetric encryption algorithm of the corresponding security strength.

Renegotiation can only be performed during the full handshake phase. However, it cannot be performed in the session resume stage with only partial handshake behavior, as the system security will be seriously affected. First, we need to set the renegotiation flag to “allow” before invoking the renegotiation. Renegotiation can operate against SSL structures. After the operation is invoked, the server requests the client to perform the renegotiation by sending a hello request message. During the second handshake, the cipher suite negotiation is completed after the server sends the server hello message. At this point, a client-aware set of secure transmission strategies is implemented. However, since the client still needs to provide the certificate for the second handshake, the two submitted certificates still need to be compared at this point. If the client refuses to submit the certificate the second time or if it is different from the first certificate, the connection will fail to handshake. Finally, the renegotiation flag needs to be banned to avoid security issues.

After the previous steps are completed, CAN based on renegotiation is fully deployed. Let us consider the design of an encryption scheme. Firstly, a default encryption scheme needs to be set for the first handshake. Because transmission requiring the highest security tends to have the highest transmission overhead, the scheme described here sets the default encryption scheme to the maximum security, so that clients that need the highest security can reduce the overhead by avoiding the overhead of renegotiation and a full handshake. When a client does not need the highest security, the renegotiation will be needed. The message sequence diagram is shown in Figure 3, in which there are four parts. Part 1 is a full handshake of two-way authentication, in which the server gets the client certificate and sets the encryption scheme. Then, in Part 2, a renegotiation is executed. Part 3 is also a full handshake, in which the server uses the new encryption scheme to negotiate with the client. Finally, Part 4 shows the data security transmission after the negotiation.

Figure 3. A message sequence diagram of client-aware negotiation based on renegotiation. The messages with “*” are for two-way authentication.

3.2. CAN Based on Redirection

Redirection means to jump from one website to another. In this manuscript, it refers to a jump from an HTTP site to an HTTPS site. This method is suitable for clients that do not support renegotiation or two-way authentication. Its overall idea is to prove the identity of the client by transmitting some information that does not require encryption and to use this client information to complete the CAN-SEAT strategy. After the server verifies the client's identity, the secure data transmission begins.

The overall process of this method is similar to the renegotiation-based method. This section briefly describes the different steps: First, the client logs in using the HTTP protocol with the identity information. The server can identify the client by the client's IP and port information, or it can retrieve the information provided by the client from the Uniform Resource Locator (URL). Then, the server sets different secure data transmission schemes according to the client's information. When a client is redirected to a connection using the HTTPS URL, a handshake is first performed, and the server compares the certificate submitted by the client at this time to the identity information in the client information table. If the user's identity information meets the criteria, the handshake completes, and the data transmission begins. For systems that do not require a client to provide a certificate, the identity of the client is not checked. The following provides a feasible example of a practical deployment.

As shown in Figure 4, we set the redirection that converts the HTTP URL to an HTTPS URL in the Apache configuration file and then specify that the clients can only log in through the HTTP URL and need to provide the client identity in the URL when logging in. After the server receives the connection request of the client, it obtains the URL when the client logs in and obtains the client identity information from the URL. Then, according to the client's identity information, the server sets the corresponding cipher suite. Clients are then redirected to sites using the HTTPS protocol. The next step is to perform a secure handshake, at which point, the client needs to set the corresponding secure transmission scheme for the client. After the first full handshake, the server also needs to match the identity from the certificate submitted by the client at this time with the previous identity information. The client is allowed to log in only if the identity information is consistent. If not, the client will fail to log in. If the server does not require a certificate, the client's identity is not detected. The secure transmission scheme is set according to the client identity information in the URL. The message sequence diagram of CAN based on redirection is similar to CAN based on renegotiation. The first two parts were replaced by three handshakes, while the last two were the same. Therefore, the message exchange procedure is not described here in too much detail.

Unlike the renegotiation-based method, which involves only SSL modules, the CAN based on redirection involves HTTP modules, as well as SSL modules. The server needs to obtain the URL in the HTTP module and set the cipher suite in the SSL module, so the method needs to coordinate the two modules. The CAN based on renegotiation can obtain the client information in the second handshake because the thread handling the two handshakes does not change. However, the method based on redirection cannot ensure that the two sessions are based on the same process, let alone the thread. This means data cannot be transferred directly between two modules. We solve this problem by storing the information table needed in the shared zone. These tables' data are also cleaned regularly to avoid increasing retrieval time due to there being too much information.

Figure 4. A flowchart of client-aware negotiation based on redirection.

4. Result and Analysis

4.1. Experimental Environment

The deployment scheme of Apache + OpenSSL was adopted in the experiment. The versions used were httpd-2.4.37 and openssl-1.1.0K, corresponding to TLSv1.2. There were three comparison platforms used. The first was the XiLinx's high-performance board card XiLinx Zynq UltraScale+ MPSoC ZCU104, which adopts the RISC instruction set and ARM architecture inside the processor. The second was the FT-2000+ processors on the prototype Tianhe-3 cluster [24], which integrates 64 cores and has a homogeneous many-core ARM architecture with a multilevel internal structure. Its operating system is YHkylin 4.0-1D. The processor is divided into eight panels that access memory individually. The third was Intel(R) Core(TM) i7-6700 CPU@3.4 GHz with the CISC instruction set and x86 architecture, whose performance is generally better than that of the ARM architecture processor, but it consumes more energy; different architectures are the main reason for the differences in energy consumption and performance. The compiler optimization instruction uses -O2.

4.2. Message Exchange

The message exchange procedure of the CAN method based on renegotiation under two-way authentication was tested. The CAN procedure based on redirection is the same as the common redirection message exchange procedure, so it was not tested here. XiLinx Zynq UltraScale+ MPSoC ZCU104 was used as the server machine. The Firefox browser (Version 50.0.1) was used to log in, and the processor of the client machine was an Intel(R) Core(TM) i7-6700.

As shown in Figure 5, if a client needs data transmission with the highest security strength, there is no need to renegotiate. After the handshake, the data transmission begins. The server hello message has the highest security strength cipher suite (Frame 8) using the symmetric encryption algorithm

AES-256-Galois Counter Mode (GCM) and the asymmetric encryption algorithm RSA. The key size of RSA is determined at the time of certificate issuance and is not reflected in the cipher suite, where 3072 bits are used.

Figure 5. The message exchange procedure of client-aware negotiation based on renegotiation with highest security strength requirement.

If a client needs data transmission without the highest security strength, renegotiation is needed. Figure 6 shows the whole message exchange procedure, which is divided into three parts: the first handshake, the server request for renegotiation, and the second handshake. The actual deployment of the secure transmission scheme occurs in the server hello message. In the server hello message of the first handshake, the cipher suite at the highest security strength is used (Frame 6). In the server hello message of the second handshake, the corresponding cipher suite is used by the client (Frame 17), after which, data transmission begins. In the second handshake, the port and IP do not change, which is the premise upon which this method can be used.

Figure 6. The message exchange procedure of client-aware negotiation based on renegotiation without the highest security strength requirement.

4.3. Encryption Efficiency

Generally, the power consumption of the processor is related to its running time. In the server cluster, the power consumption can be reduced by shutting down the servers. In WSNs,

energy consumption can be reduced by using sleeping computing nodes. If data processing and transmission cost can be reduced, then the above two methods can be used to indirectly reduce energy consumption. Therefore, this section indirectly reflects the contribution of this method to energy consumption reduction by measuring data processing and transmission time. Here, the data processing time refers to the time required to prepare the secure data to be sent on the server side.

The data processing overhead ($T_t(m)$) can be divided into three parts, namely asymmetric encryption overhead (T_a), symmetric encryption overhead ($T_s(m)$), and the rest overhead (o). Symmetric encryption overhead is proportional to the amount of data transferred (m). The asymmetric encryption overhead is constant. However, the rest overhead can be treated as a fixed value because it is variable, but much smaller than the other two parts. When the overhead of the CAN-SEAT strategy is not taken into account, the expression of $T_t(m)$ is as follows:

$$T_t(m) = o + T_a + T_s(m) \quad (1)$$

When the overhead of the CAN-SEAT strategy is taken into account ($T_c(m)$), the overhead of the second full handshake and renegotiation is considered. However, the overhead of renegotiation is so little that it can be taken into account in the rest overhead. Now, the expression of $T_c(m)$ is as follows:

$$T_c(m) = o + 2 \times T_a + T_s(m) \quad (2)$$

In the process of secure data transmission, the main overhead of the processor comes from asymmetric and symmetric encryption and decryption. We compared the efficiency of encryption and decryption to analyze the contribution of the CAN-SEAT strategy to the actual system. The asymmetric encryption and decryption algorithm was tested using the RSA algorithm, which can be used in key exchange and authentication. In asymmetric encryption, the speed of the encryption is called the signature speed, whereas the speed of the decryption is called the verification speed. The encryption efficiency is described by the number of times the algorithm is executed per second. Symmetric encryption and decryption were tested by AES and ChaCha20-Poly1305, and the encryption efficiency is described by the amount of encrypted data per unit time. As not all machines support AES hardware instructions, the use of AES hardware instructions was analyzed in the experiment. As the block size of symmetric encryption affects the encryption efficiency, we only tested the block size (16 bytes) actually used in OpenSSL, and the experimental results are shown in Figure 7. The signature algorithm (such as SHA) is often used in these algorithms, so we did not test the encryption efficiency of the signature algorithm.

The ratio of the lowest security strength to the highest security strength in the experimental results is shown in Table 2. In the experiment, the highest strength key size of the RSA algorithm was 7680 bits, and the lowest was 512 bits. Figure 7a,c,e compares the signature and authentication efficiency of the asymmetric encryption algorithm. As the RSA algorithm key size increases, the encryption efficiency of the asymmetric encryption algorithm decreases. If the CAN-SEAT strategy is adopted, the overhead of the asymmetric encryption can be considerably reduced. For systems with less data transmitted in a fixed time, the time spent for data processing and transmission can be markedly reduced as well.

Figure 7b,d,f compares the encryption efficiency of AES and the stream encryption algorithm ChaCha20-Poly1305 [25], where AES New Instructions (AES-NI) denotes AES using hardware instruction, AES denotes AES without hardware instruction, and Cipher Block Chaining (CBC) is the block encryption method that is still widely used at present. ChaCha20-poly1305 is a widely-used stream encryption algorithm with 256 bit symmetric encryption strength, and the strength of other key sizes is not implemented in OpenSSL, so it could not be tested. Process performance is improved using AES hardware instructions, but not all machines support AES hardware instructions; these machines can only use software algorithms. Symmetric encryption is used to encrypt the transmitted data and is the most time consuming part of encryption for systems that need to transmit a large amount of data in a fixed period of time.

Figure 7. Comparison of encryption efficiency of the asymmetric encryption and the symmetric encryption algorithm in these platforms: Xilinx Zynq UltraScale+, FT-2000+: FTC662, and Intel Core i7-6700. (a) RSA in Xilinx Zynq; (b) AES in Xilinx Zynq; (c) RSA in FT-2000+: FTC662; (d) AES in FT-2000+: FTC662; (e) RSA in Intel Core i7-6700; (f) AES in Intel Core i7-6700.

Compared with the AES algorithm, which does not use hardware instruction, ChaCha20-Poly1305 has higher encryption efficiency on these processors. Therefore, the ChaCha20-Poly1305 algorithm can be used against the client when the client does not support AES hardware instructions. When the version of the client's SSL/TLS protocol is older, only the AES algorithm can be used at this point. For clients that do not support AES hardware, only AES software encryption can be used on three types of processors. However, for processors that support AES hardware instructions, the AES algorithm is the preferred option over ChaCha20-Poly1305. As shown in Figure 7, the use of AES hardware instructions can yield improved performance for different block algorithms. In practice, if the client

supports the AES hardware instruction, different AES encryption strengths can be set according to the client's identity information.

Table 2. The ratio of the lowest security strength to the highest security strength encryption overhead in the experiment.

Experimental Platform	Xilinx Zynq	FT-2000+:FTC662	Intel I7-6700
RSA signature speed	3.07%	2.13%	2.20%
RSA authentication speed	8.79%	7.35%	9.95%
AES software encryption speed	76.08%	76.99%	74.48%
AES hardware encryption speed	87.12%	95.48%	87.49%

4.4. Encryption Scheme and Security Analysis

In the SSL/TLS protocol, cipher suites are used to handle encryption schemes. Two common cipher suites, ECDHE-RSA-AES256-GCM-SHA384 and ECDHE-RSA-AES128-GCM-SHA256, with different security strengths are used to illustrate. The cipher suite describes the algorithm of key exchange (ECDHE), the algorithm of authentication (RSA), data encryption (AES128, AES256), and signature (SHA384, SHA256) used in the current session. The algorithm strength of symmetric encryption is shown in Figure 7b,d,f. In addition, there is a factor that affects the encryption overhead, that is the algorithm and strength of asymmetric encryption used by the certification authority when issuing certificates to clients. The encryption efficiency of asymmetric encryption algorithms is shown in Figure 7a,c,e. Using different algorithms (such as ECDSA or RSA) or different algorithm strengths will result in different security and encryption efficiency. The algorithm strengths are shown in Table 1.

However, sometimes, in order to reduce the overhead of the full handshake, the encryption scheme used by each IP can be stored locally. In this way, the next time the client logs in, because the IP is available in the client hello message and the cipher suite is set after this message, the cipher suite setting can be completed without the second handshake. At this time, there will be no overhead for the second handshake. If the client needs to modify the encryption scheme, they can also perform a second handshake to change the encryption scheme.

Therefore, the CAN based on renegotiation will only generate some small processing overhead in some cases (the method described in the previous paragraph), and in some cases, it will generate the overhead of a full handshake. The redirection-based method will produce a three-way handshake overhead. Because usually the time for data caching (without deleting session-related information) will last for several minutes, the overhead of a handshake during this time is usually negligible. Therefore, the CAN-SEAT strategy can assume that there is no overhead in most cases. However, there are still situations where it is difficult to ignore the overhead. At this time, a specific method may be designed according to the specific situation, such as using the method described in the previous paragraph to avoid the overhead of CAN-SEAT. In this way, it can be considered that the proposed strategy has no overhead in most cases.

The security analysis mainly is for the security of the encryption algorithm and the security of the two CAN-SEAT strategies. The strength of the algorithm is shown in Section 2.3. In the actual deployment here, only algorithms with the strength of the near-term or long-term protection algorithm (Table 1) will be deployed. Therefore, the strength of the algorithm can be guaranteed. As for the second strategy, in the first stage of CAN based on redirection, only the client identity information is need. Other relevant private information is not transmitted over the network and is therefore secure, and the HTTPS protocol based on Apache + OpenSSL will be used in the later stage, while the continuously maintained security software ensures the most secure communication environment in today's environment. The techniques used by CAN based on renegotiation mainly include renegotiation. There are some weaknesses in the renegotiation technology, so in the latest version of OpenSSL, there have been several security updates to the renegotiation technology [26,27].

The actual system deployed here uses the latest version of the software, so it is relatively safe in the current environment.

4.5. Application

In WSNs, data need to be transmitted periodically. To ensure the integrity and security of data transmission, encryption transmission is often required; the amount of data transmitted at this time tends to be not large. In the process of transmitting encrypted data using the browser/server architecture, the amount of data transferred is usually less than 10 MB, so we simulated actual application scenarios with four sizes: 10 KB, 100 KB, 1 MB, and 10 MB.

We built both servers and clients on the same Intel Core i7-6700 processor to avoid transmission delays. In this experiment, the client sent the same data several times and the server cleared the data inside the SSL structure after each reception to avoid the influence of session resume. Finally, the server calculates the average receiving time to describe the data processing and transmission times. To accurately describe the processing overhead in the actual application, the client starts by reading the file, and the receiver writes the received data to the file. Notably, for the server, the receive time refers to the time from the handshake until the data are fully written to the file. In this way, the actual cost consists of three parts: handshake time, transfer time, and time to write file. If there is no write time, better experimental results can be obtained because with the same data size, the write time is fixed, which reduces the improvement effect.

The actual data processing and transmission times for applications using Intel Core i7-6700 processor are shown in Table 3, with similar results for the three remaining processors. The actual processing and transmission times of the four sizes of data in four scenarios were tested, including the time of handshake, file IO processing, and encrypted data transmission. Among them, symmetric encryption adopted the AES algorithm in Galois Counter Mode (GCM) mode (AEAD), whereas asymmetric encryption adopted the RSA algorithm, whose security strength is reflected in the key size of the algorithm. Specifically, cipher suite ECDHE-RSA-AES256-GCM-SHA384 was adopted in Scenarios 1 and 3, where the key size of the RSA algorithm was 4096 bits, that of the AES algorithm was 256 bits, and that of the SHA algorithm was 384 bits. Scenarios 2 and 4 adopted the cipher suite ECDHE-RSA-AES128-GCM-SHA256, where the key sizes of RSA, AES, and SHA algorithms were 1024, 128, and 256 bits, respectively. Scenarios 1 and 2 adopted software encryption; Scenarios 3 and 4 adopted hardware encryption. In the case of software encryption, the data processing and transmission times can be reduced by 89.41%, 82.43%, 54.08%, and 15.40%, respectively, whereas in the case of hardware encryption, the data processing and transmission times can be reduced by 85.30%, 81.90%, 61.62%, and 24.63%, respectively. In general, the less the amount of data transmitted, the greater the benefit of the CAN-SEAT strategy.

The above experience did not consider the overhead brought by CAN-SEAT, because this overhead can usually be avoided by design. However, it is inevitable that there are some situations where the overhead cannot be avoided. Therefore, we forced the use of the CAN based on renegotiation for the above four scenarios to determine how to design the scheme and the benefits of using this strategy when the overhead of the CAN-SEAT strategy cannot be avoided. The results are shown in Table 4. We can compare the two tables to get the overhead of the CAN-SEAT strategy. When the amount of data is small (10KB), the overhead ratio is about 50%. When the data volume is large (10MB), the proportion can reach 16.7% at most. However, when comparing Scenarios 2 and 4 in Table 4 with Scenarios 1 and 3 in Table 3, using the CAN-SEAT method can always bring efficiency improvements. In the case of software encryption, the data processing and transmission times can be reduced by 77.21%, 72.26%, 49.11%, and 14.45%, respectively, whereas in the case of hardware encryption, the data processing and transmission times can be reduced by 69.68%, 64.28%, 48.84%, and 21.87%, respectively.

Table 3. Total data processing and transmission times on the Intel processor (μ s).

File Size	10 KB	100 KB	1 MB	10 MB
Scenario 1	7410	8289	15,025	78,805
Scenario 2	785	1456	6900	66,672
Scenario 3	5027	5259	6969	28,137
Scenario 4	739	952	2675	21,209

Table 4. Total data processing and transmission times on the Intel processor (μ s) when the CAN-SEAT strategy is forced to be used.

File Size	10 KB	100 KB	1 MB	10 MB
Scenario 1	15,105	15,770	22,478	86,297
Scenario 2	1689	2299	7646	67,415
Scenario 3	10,150	10,412	12,021	32,836
Scenario 4	1524	1878	3565	21,984

5. Conclusions

Aiming at the problem that the security and efficiency cannot be considered in the process of data transmission, this paper proposes a strategy called Client-Aware Negotiation for Secure and Efficient Data Transmission (CAN-SEAT), so that clients with different security transmission requirements can use the appropriate data security transmission scheme without modifying the client. Considering the diversity of clients in the actual deployment process, two implementation methods were proposed based on renegotiation and redirection, separately. Then, a specific implementation scheme based on Apache + OpenSSL was described. The CAN-SEAT strategy was tested under four typical application scenarios, and the data processing and transmission time were reduced by 89.41% in the optimal case, while the effectiveness of the strategy was verified on multiple platforms. With the continuous update of the SSL/TLS protocol, determining how to smoothly apply the CAN-SEAT strategy to the new protocol needs to be further studied in the implementation mode.

Author Contributions: Conceptualization, Z.W. and H.C.; methodology, Z.W. and H.C.; software, Z.W.; validation, Z.W., H.C., and W.W.; formal analysis, Z.W. and H.C.; investigation, Z.W. and H.C.; writing, original draft preparation, Z.W.; writing, review and editing, Z.W. and H.C.; visualization, Z.W.; supervision, H.C.; project administration, W.W. All authors read and agreed to the published version of the manuscript.

Funding: This authors gratefully acknowledge the financial support from the National Key Research and Development Plan of China (No. 2016YFB0201800).

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

References

1. Mehmood, A.; Mauri, J.L.; Noman, M.; Song, H. Improvement of the Wireless Sensor Network Lifetime Using LEACH with Vice-Cluster Head. *Ad Hoc Sens. Wirel. Netw.* **2015**, *28*, 1–17.
2. Kuila, P.; Jana, P.K. Energy efficient clustering and routing algorithms for wireless sensor networks: Particle swarm optimization approach. *Eng. Appl. Artif. Intell.* **2014**, *33*, 127–140, [[CrossRef](#)]
3. Wei, Y.; Ma, X.; Yang, N.; Chen, Y. Energy-Saving Traffic Scheduling in Hybrid Software Defined Wireless Rechargeable Sensor Networks. *Sensors* **2017**, *17*, 2126, [[CrossRef](#)]
4. Liu, X.; Fu, L.; Wang, J.; Wang, X.; Chen, G. Multicast Scaling of Capacity and Energy Efficiency in Heterogeneous Wireless Sensor Networks. *ACM Trans. Sens. Netw.* **2019**, *15*, 1–32, [[CrossRef](#)]
5. Ebrahimi, D.; Assi, C. Network Coding-Aware Compressive Data Gathering for Energy-Efficient Wireless Sensor Networks. *ACM Trans. Sens. Netw.* **2015**, *11*, 1–24, [[CrossRef](#)]
6. Yu, X.; Baek, S.J. Energy-Efficient Collection of Sparse Data in Wireless Sensor Networks Using Sparse Random Matrices. *ACM Trans. Sens. Netw.* **2017**, *13*, 1–36. [[CrossRef](#)]

7. Qiu, Y.; Li, S.; Xu, X.; Li, Z. Talk more listen less: Energy-efficient neighbor discovery in wireless sensor networks. In Proceedings of the 35th Annual IEEE International Conference on Computer Communications (INFOCOM), San Francisco, CA, USA, 10–14 April 2016; pp. 1–9, [CrossRef]
8. Yin, C.; Jin, S. An energy-saving strategy based on multi-server vacation queuing theory in cloud data center. *J. Supercomput.* **2018**, *74*, 6766–6784, [CrossRef]
9. Kuehn, P.J.; Mashaly, M.E. Automatic energy efficiency management of data center resources by load-dependent server activation and sleep modes. *Ad Hoc Netw.* **2015**, *25*, 497–504, [CrossRef]
10. Nguyen, B.M.; Tran, D.; Nguyen, G. Enhancing service capability with multiple finite capacity server queues in cloud data centers. *Clust. Comput.* **2016**, *19*, 1747–1767, [CrossRef]
11. Joseph, S.; Hao, Z.; Pasi, E.; Hannes, T. Transport Layer Security (TLS) Session Resumption without Server-Side State. *RFC* **2008**, *5077*, 1–20, [CrossRef]
12. Cremers, C.; Horvat, M.; Hoyland, J.; Scott, S.; van der Merwe, T. A Comprehensive Symbolic Analysis of TLS 1.3. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS), Dallas, TX, USA, 30 October–3 November 2017; Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D., Eds.; ACM: New York, NY, USA, 2017; pp. 1773–1788, [CrossRef]
13. Gergely, A.; Lejla, B.; Lynn, M.B.; Veelasha, M.; Antoine, G.; Iynkaran, N. New directions in IoT privacy using attribute-based authentication. In Proceedings of the ACM International Conference on Computing Frontiers (CF'16), Como, Italy, 16–19 May 2016; ACM: New York, NY, USA, 2016; pp. 461–466, [CrossRef]
14. Tanesh, K.; An, B.; Lynn, M.B.; Madhusanka, L.; Mika, Y. Identity privacy preserving biometric based authentication scheme for Naked healthcare environment. In Proceedings of the IEEE International Conference on Communications (ICC 2017), Paris, France, 21–25 May 2016; pp. 461–466, [CrossRef]
15. António, P.; Ricardo, C. Hash-Chain Based Authentication for IoT Devices and REST Web-Services. In *Advances in Intelligent Systems and Computing, Proceedings of the Ambient Intelligence—Software and Applications-7th International Symposium on Ambient Intelligence (ISAmI 2016), Seville, Spain, 1–3 June 2016*; Springer: Berlin/Heidelberg, Germany, 2016; pp. 189–196. [CrossRef]
16. Gai, K.; Qiu, L.; Chen, M.; Zhao, H.; Qiu, M. SA-EAST: Security-Aware Efficient Data Transmission for ITS in Mobile Heterogeneous Cloud Computing. *ACM Trans. Embed. Comput. Syst.* **2017**, *16*, 60:1–60:22, [CrossRef]
17. Shang, X.; Liu, A.; Wang, Y.; Xie, Q.; Wang, Y. Energy-Efficient Transmission Based on Direct Links: Toward Secure Cooperative Internet of Things. *Wirel. Commun. Mob. Comput.* **2018**, *2018*, 5012096, [CrossRef]
18. Zhang, M.; Cumanan, K.; Wang, W.; Burr, A.G.; Ding, Z.; Lambbotharan, S.; Dobre, O.A. Energy Efficiency Optimization for Secure Transmission in a MIMO-NOMA System. In Proceedings of the 2020 IEEE Wireless Communications and Networking Conference (WCNC), Seoul, Korea, 25–28 May 2020; pp. 1–6, [CrossRef]
19. Wang, D.; Bai, B.; Chen, W.; Han, Z. Energy efficiency maximization for secure data transmission over DF relay networks. In Proceedings of the 2015 IEEE International Conference on Communications (ICC), London, UK, 8–12 June 2015; pp. 2313–2317, [CrossRef]
20. Netcraft Ltd. June 2020 Web Server Survey. Available online: <https://news.netcraft.com/archives/category/web-server-survey> (accessed on 18 July 2020).
21. Bellare, M.; Tackmann, B. The Multi-user Security of Authenticated Encryption: AES-GCM in TLS 1.3. In *Lecture Notes in Computer Science, Proceedings of the Advances in Cryptology—CRYPTO 2016: 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, 14–18 August 2016*; Robshaw, M., Katz, J., Eds.; Springer: Berlin/Heidelberg, Germany, 2016; Volume 9814, pp. 247–276. [CrossRef]
22. Abdalla, M.; Gierlichs, B.; Paterson, K.G.; Rijmen, V.; Sadeghi, A.R.; Smart, N.P.; Stam, M.; Ward, M.; Warinschi, B.; Watson, G. Algorithms, Key Size and Protocols Report. Available online: <https://www.ecrypt.eu.org/csa/documents/D5.4-FinalAlgKeySizeProt.pdf> (accessed on 18 July 2020).
23. Giry, D. BlueKrypt-Cryptographic Key Length Recommendation: ECRYPT-CSA Recommendations. Available online: <https://www.keylength.com/en/3> (accessed on 18 July 2020).
24. You, X.; Yang, H.; Luan, Z.; Liu, Y.; Qian, D. Performance Evaluation and Analysis of Linear Algebra Kernels in the Prototype Tianhe-3 Cluster. In *Lecture Notes in Computer Science, Proceedings of the Supercomputing Frontiers: 5th Asian Conference (SCFA 2019), Singapore, 11–14 March 2019*; Abramson, D., de Supinski, B.R., Eds.; Springer: Berlin/Heidelberg, Germany, 2019; Volume 11416, pp. 86–105. [CrossRef]
25. Nir, Y.; Langley, A. ChaCha20 and Poly1305 for IETF Protocols. *RFC* **2018**, *8439*, 1–46, [CrossRef]

26. Eric, R.; Marsh, R.; Steve, D.; Nasko, O. Transport Layer Security (TLS) Renegotiation Indication Extension. *RFC* **2010**, *5746*, 1–15, [[CrossRef](#)]
27. Florian, G.; Florian, K.; Douglas, S. On the Security of TLS Renegotiation. In Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security (CCS'13), Berlin, Germany, 4–8 November 2013; Ahmad-Reza, S., Virgil, D., Moti, Y., Eds.; ACM: New York, NY, USA, 2013; pp. 387–398, [[CrossRef](#)]

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).