Hindawi
Security and Communication Networks
Volume 2017, Article ID 9479476, 15 pages
https://doi.org/10.1155/2017/9479476

*Research Article*

# Enc-DNS-HTTP: Utilising DNS Infrastructure to Secure Web Browsing

**Mohammed Abdulridha Hussain,[1,2] Hai Jin,[1] Zaid Alaa Hussien,[1,3] Zaid Ameen Abduljabbar,[1,2] Salah H. Abbdal,[1] and Ayad Ibrahim[2]**

[1]*Cluster and Grid Computing Lab, Services Computing Technology and System Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China*
[2]*University of Basrah, Basrah, Iraq*
[3]*Southern Technical University, Basrah, Iraq*

Correspondence should be addressed to Hai Jin; hjin@hust.edu.cn

Received 6 August 2016; Revised 31 October 2016; Accepted 15 November 2016; Published 3 April 2017

Academic Editor: Pascal Lorenz



Online information security is a major concern for both users and companies, since data transferred via the Internet is becoming increasingly sensitive. The World Wide Web uses *Hypertext Transfer Protocol* (HTTP) to transfer information and *Secure Sockets Layer* (SSL) to secure the connection between clients and servers. However, *Hypertext Transfer Protocol Secure* (HTTPS) is vulnerable to attacks that threaten the privacy of information sent between clients and servers. In this paper, we propose Enc-DNS-HTTP for securing client requests, protecting server responses, and withstanding HTTPS attacks. Enc-DNS-HTTP is based on the distribution of a web server public key, which is transferred via a secure communication between client and a *Domain Name System* (DNS) server. This key is used to encrypt client-server communication. The scheme is implemented in the C programming language and tested on a Linux platform. In comparison with Apache HTTPS, this scheme is shown to have more effective resistance to attacks and improved performance since it does not involve a high number of time-consuming operations.

## 1. Introduction

Digital information and electronic services are delivered to users through the Internet. Information and services are often sensitive, particularly in applications such as online banking, which requires secure transactions [1]. Services are provided through a web server, and the user contacts the server by using an Internet browser such as Internet Explorer (IE) or Google Chrome. The client and server communicate using *Hypertext Transfer Protocol* (HTTP).

Modern web security relies on a combination of *Secure Sockets Layer*/*Transport Layer Security* (SSL/TLS) with HTTP, and this is known as *Hypertext Transfer Protocol Secure* (HTTPS). However, almost all browsers and servers apply SSL/TLS to secure information transactions [2].

SSL uses server certificates to publish the public keys of server; each of these certificates is signed by a trusted third party, known as the *Certificate Authority* (CA). SSL protocol consists of two phases, the handshake phase and the data transfer phase, and the details of these are discussed further in Section 2. The handshake phase includes the sharing of both the server certificate and a symmetric algorithm key. Handshake messages are sent in plain text until the server successfully transmits the certificate to the client. In this context, data transfer is protected by a symmetric cipher.

These plain text messages in the handshake phase are particularly targeted by attackers, threatening HTTPS security. Attackers exploit a loophole by impersonating the web server and stealing user information; this approach capitalises on the user's habit of typing a *Uniform Resource Locator* (URL) without specifying HTTPS. Attackers deceive the client by making it appear that the web server is using HTTP without security. Such attacks are known as *Man-in-the-Middle* (MITM) sniffing and stripping attacks.

In simple terms, a web server is identified by the web browser using *Internet Protocol* (IP). The browser obtains the server's IP address from a *Domain Name System* (DNS) server by sending a DNS query, which contains the *domain name* (DN). The DNS server replies with the server IP, and the browser program then sends an HTTPS request with the destination IP set to this server IP. In the interests of safety, HTTPS secures client-server communication by encrypting HTTP. However, attackers can take advantage of the unsecured DNS communication by spoofing these DNS messages and replacing the server IP with the attacker's IP. The web browser then sends an HTTPS message to the attacker's IP as the destination server. If the attacker deploys a website to answer the client request, then the attack is undetected. Such attacks are known as DNS spoofing or phishing [2].

The main contributions of this paper to network security are as follows:

(i) We address the question of how to protect the privacy of the user while browsing the Internet. To this end, we present the Enc-DNS-HTTP scheme which protects a user surfing the Internet from attacks by encrypting both DNS and HTTP messages using asymmetric and symmetric cipher algorithms.

(ii) The proposed scheme can resist MITM disclosure sniffing and stripping attacks against the communication between the client and the web server.

(iii) The proposed scheme prevents an attacker from modifying the real web server IP through a DNS attack.

(iv) The scheme is applicable to and compatible with current Internet hardware infrastructure. Within this context, DNS and HTTP messages are maintained without affecting functionality.

(v) In terms of computational cost, the scheme is shown to outperform HTTPS for high performance value.

The remainder of this paper is structured as follows: Section 2 provides preliminary and contextual information; Section 3 defines the problem addressed in this work; Section 4 describes prior work related to this issue; Section 5 explains the proposed scheme; Section 6 presents the implementation of the Enc-DNS-HTTP scheme in detail; Section 7 presents the experiments carried out to verify this scheme; Section 8 presents the results of experiment and a discussion; Section 9 explains the security analysis carried out; and the final section presents the conclusion.

## 2. Preliminaries

### 2.1. Notations

*2.1.1. Entities.* Network communication consists of a *client* ($C$), *web server* (WS), *DNS server* (DS), and *attacker* (ATT). When masquerading as a WS, the attacker is denoted as ATT[WS]; when masquerading as a $C$, the attacker is denoted as ATT[$C$]. All entities are connected through the Internet, except when ATT is on the same *Local Area Network* (LAN) as either $C$ or $S$. The Internet gateway is the *router* ($R$).

The *Browser Authority* (BA) is the creator of the client-side browser program, which has both private and public keys. The public key is distributed to the client by the browser.

*2.1.2. Exchanged Messages*

(i) Msg represents a random message. For example, a term that $C$ sends as a Msg to WS is denoted as follows:

$$C \longrightarrow \text{WS: Msg.} \tag{1}$$

(ii) The term that $C$ sends as a Msg to WS called $Y$ and containing $Z$ is denoted as follows:

$$C \longrightarrow \text{WS: } Y(Z). \tag{2}$$

*2.1.3. Parameters.* A parameter generated by $C$ is expressed as $X_C$, and this means that the $X$ parameter is created by $C$. The parameter notation is shown in Notations section.

*2.1.4. Functions.* The function notation is shown in Notations section. It should be noted that symmetric ciphers use the same key for different functions, such as encryption and decryption, whereas an asymmetric cipher uses public or private keys for the same functions of encryption and decryption. For simplicity, we distinguish asymmetric encryption and decryption as we would symmetric encryption and decryption, in which each uses a separate key.

*2.2. HTTPS.* Internet privacy is provided by securing the HTTP connections between web browsers and web servers; this is known as HTTPS.

*2.2.1. HTTP.* HTTP is used to access data on the *World Wide Web* (WWW) and is used as the protocol for data communication. HTTP transactions use the services of *Transmission Control Protocol* (TCP) on port 80, where the primary HTTP messages are requested and responses are received between client and server. HTTP does not support security, and is a stateless protocol [3].

*2.2.2. SSL/TLS.* The SSL protocol and its successor, the TLS protocol, are standardised protocol suites which were introduced by Netscape in 1995 to provide secure communication between a client and server over an insecure network. SSL/TLS protects information transmission using a symmetric cipher with a key, whose key component is shared by an asymmetric cipher. Server authentication is accomplished via digital certificate by applying X.509 technology [4, 5].

SSL/TLS achieves confidentiality by using encryption; it ensures integrity through the use of a message authentication code, which is a hash function with a key, and authentication by a digital certificate. A client can validate a server certificate using the CA's public key, which is possessed by the client and typically preinstalled into the web browser [6].

SSL/TLS is composed of two layers: the upper layer, which contains the Handshake protocol for establishing a session

Table 1: Establishing a session using the handshake protocol.

| | | |
|---|---|---|
| (M1) | CB → WS: | ClientHello ($CSL_C$, $S\_ID_C$, $RN_C$) |
| (M2) | WS → CB: | ServerHello ($CSC_{WS}$, $S\_ID_{WS}$, $RN_{WS}$) |
| (M3) | WS → CB: | Certificate ($PK_{WS}$, sig) |
| | $C$: Generate $X_C$ random number; $Y_C = \text{Enc}(X_C; PK_{WS})$; $SK_C = H(X_C \,\|\|\, RN_C \,\|\, RN_{WS})$ | |
| (M4) | CB → WS: | ClientKeyExchange ($Y_C$) |
| | $S$: $X_C = \text{Enc}(Y_C; RK_{WS})$; $SK_{WS} = H(X_C \,\|\|\, RN_C \,\|\, RN_{WS})$ | |
| (M5) | CB → WS: | ChangeCipherSpec () |
| (M6) | CB → WS: | Finished(Enc($H(SK_C \,\|\|\, \text{Msg\_all})$; $SK_C$)) |
| (M7) | WS → CB: | ChangeCipherSpec () |
| (M8) | WS → CB: | Finished(Enc($H(SK_{WS} \,\|\|\, \text{Msg\_all})$; $SK_{WS}$)) |

and the Alert protocol for communicating error messages and application protocols; and the lower layer, which includes the Record protocol for exchanging messages using current connection parameters, obtained from the upper layer [5, 7].

The simplest scenario for establishing a session using SSL is shown in Table 1, where messages (M1) and (M2) define the first phase of SSL/TLS. The first phase is cipher suite negotiation, which sets and shares the symmetric cipher key parameters $RN_C$ and $RN_{WS}$. Message (M3) is the server certificate that contains the server public key.

The second phase is symmetric cipher key construction, which is initiated by (M4). Messages (M5) and (M7) communicate that all further messages will now be encrypted by using the key and the symmetric cipher previously agreed upon. The final phase is secure transmission, assuming that the nodes manage to decrypt the finished messages. The Record protocol continues after the final phase until the session is terminated or destroyed [8–10].

*2.3. DNS.* DNS is a distributed database that translates DNs into IP addresses. The TCP/IP suite uses an IP address to route packets; however, the host name is more appropriate for human readability [11, 12].

As DNs must be globally unique, a hierarchical name space is used as the DN space; this is designed in the form of a tree structure, with the root at the top. Each node in the tree has a label, which is a string of characters, although the root label is an empty string. However, DN is a sequence of labels separated by dots from the node up to the root.

The DN space is distributed across numerous computers known as DNS servers. The division of DN space is based on the domain, which is a subtree of the DN space, and is sometimes known as the zone. The name of the domain is the name of the node at the top of the subtree [3].

DNS servers store domain information in a data structure known as a *Resource Record* (RR). Each RR has an associated name, class, and type. A *Resource Record Set* (RRset) describes a situation in which multiple RRs are associated with the same name; in this case, the domain has more than one IP [11].

DNS is designed as a client-server application. Within the client-side application of DNS, a resolver receives the DN from the browser and sends a mapped request query to the DNS server. The two types of DNS messages are described below [3]:

(i) DNS query: the resolver creates a DNS query that contains an *identifier* (ID). The ID differs for each message and port number. The Question section of the DNS query is filled with the DN.

(ii) DNS response: the server creates a DNS response; this contains the same ID in order to identify the query. The Question section contains the DN, while the Answer section contains the IP, authoritative section, and additional information section.

DNS messages are sent without encryption or authentication, thereby increasing the risk of attack. DNS is vulnerable to spoofing and cache poisoning attacks, which are intended to redirect client traffic to an attacker's machine or a fake website [13–15].

## 3. Problem Definition

*3.1. Internet Model.* Internet browsing requires three nodes: the web server, the DNS server, and the client. The web server hosts the web page as a service to the client. The DNS server maps the host name to the IP, as described in Section 2. The client is the user's interface to the Internet, which runs the web browser program to handle user requests and server responses. Web browser programs such as IE and Chrome are created by companies like Microsoft and Google, for example. In this work, the company which created the browser will be referred to as the *Browser Authority*.

Each client connects to the Internet through a router, which provides the client with an $IP_C$, DNS $IP_{DS}$, and the router's own $IP_R$ as a gateway. Internet surfing typically begins when the user enters the URL in the CB; the browser program then carries out the following steps.

*Step 1.* CB fetches DN from URL and submits DN to DNS resolver.

*Step 2.* $C$ sends *Address Resolution Protocol* (ARP) request with gateway $IP_R$.

Table 2: DNS spoofing attack.

| | | | |
|---|---|---|---|
| ATT within $C$ LAN legitimately; ATT must reside between $C$ and $R$ | | | |
| (M1) | ATT $\rightarrow C$: | ARP_Reply $(\text{IP}_R, \text{MAC}_{\text{ATT}}) \times n$ | ARP poisoning |
| (M2) | ATT $\rightarrow R$: | ARP_Reply $(\text{IP}_C, \text{MAC}_{\text{ATT}}) \times n$ | |
| The result is ATT[$R$] for $C$ and ATT[$C$] for $R$ | | | |
| (M3) | $C \rightarrow$ ATT[$R$]: | DNS_Query ($\text{ID}_1$, DN); dest. IP = $\text{IP}_{\text{DS}}$ | DNS spoofing |
| (M4) | ATT[$C$] $\rightarrow R \rightarrow$ DS: | DNS_Query ($\text{ID}_1$, DN); dest. IP = $\text{IP}_{\text{DS}}$ | |
| (M5) | DS $\rightarrow R \rightarrow$ ATT[$C$]: | DNS_Reply ($\text{ID}_1$, DN, $\mathbf{IP_{WS}}$); dest. IP = $\text{IP}_C$ | |
| (M6) | ATT[$R$] $\rightarrow C$: | DNS_Reply ($\text{ID}_1$, DN, $\mathbf{IP_{ATT}}$); dest. IP = $\text{IP}_C$ | |
| All client HTTP traffic is sent to ATT directly | | | |
| (M7) | $C \rightarrow$ ATT[WS]: | HTTP_Request (URL); dest. IP = $\text{IP}_{\text{ATT[WS]}}$ | |
| (M8) | ATT[WS] $\rightarrow C$: | HTTP_Response (HTML); dest. IP = $\text{IP}_C$ | |

*Step 3.* $R$ responds with its $\text{MAC}_R$ to $C$.

*Step 4.* DNS resolver creates a DNS query containing DN and sends the query in LAN using $\text{MAC}_R$ to the Internet using $\text{IP}_{\text{DS}}$.

*Step 5.* $C$ receives DNS reply from DS through $R$ carrying $\text{IP}_{\text{WS}}$.

*Step 6.* The resolver delivers $\text{IP}_{\text{WS}}$ to CB.

*Step 7.* CB creates HTTP request containing URL and sends the request through LAN using $\text{MAC}_R$ and through the Internet using $\text{IP}_{\text{WS}}$.

*Step 8.* WS responds to $C$ with the service requested, such as an HTML page.

All DNS and HTTP messages are transferred in plain text without the use of an encryption process to maintain security, and this makes the model vulnerable to various types of attack.

*3.2. Statement of the Problem.* The problem is how to secure web browsing, or rather, how to secure message transfers in the Internet model. A trivial solution is the use of SSL/TLS to secure HTTP traffic, referred to as HTTPS. However, this model is vulnerable to various attacks, and several of these are described further in Section 9.1.

Unfortunately, SSL secures HTTP and leaves DNS unsecured; this can then be exploited by DNS spoofing attacks if ATT is in the same LAN as $C$ (Table 2), or by DNS poisoning attack if ATT is on the Internet (Table 3).

During the browsing process, even if HTTP transacts securely using SSL, the MITM attack finds a loophole in the protocol by sniffing and stripping attacks. DNSSEC [16] has been proposed in place of SSL for securing DNS messages and the Internet; however, this has not been employed due to its complexity and poor performance, as discussed further in Section 4.1.

## 4. Related Work

Protecting web browsers from attackers has attracted a great deal of research attention, due to the plethora of data transactions and information on the Internet. SSL/TLS has been deployed for securing HTTP transactions and has been the focus of both developers and attackers worldwide. SSL vulnerabilities have been known for several years, and several suggestions have emerged in the literature for replacing, modifying, or complementing SSL/TLS. The primary approaches which have been proposed for web security can be categorised as follows.

*4.1. Utilising DNS.* The approaches in this category secure the web through the use of the DNS server, which is part of the infrastructure of the Internet. DNSSEC was created to secure DNS messages and to protect the web from major attacks such as DNS spoofing and has been proposed for use in sharing web server public keys. However, DNSSEC suffers from certain problems which have hindered its wider adoption [11].

DNSSEC uses various signature algorithms and hash functions; however, no standard has been agreed on for the specification of a single algorithm or function. Consequently, the DNS server sends to the client resolver a response containing all the keys and signatures which are supported; this results in higher consumption of bandwidth and processing time and lower performance. The authors of [17] have proposed a cipher suite negotiation which selects the strongest algorithm from the DNS server list as the negotiation parameter. This is then sent in plain text, which places the communication at risk of a MITM attack which tricks both client and server into using a weak algorithm by changing the client list, after which it would be straightforward for the attacker to break the security.

DANE [18] and CAA [19] are IETF standards that propose the use of DNS infrastructure to validate web server certificates. DANE is a method based on the use of DNSSEC, while CAA may use DNSSEC as an option; DANE is therefore affected by all the vulnerabilities of DNSSEC. Both standards

Table 3: DNS cache poisoning attack.

| | | |
|---|---|---|
| ATT not in $C$ LAN; ATT must redirect the traffic to his machine. ATT opportunity: if local DS does not have $IP_{WS}$, then the request is sent to zone server: | | |
| (M1) | $C \rightarrow DS_L$: | DNS_Query ($ID_1$, DN); dest. IP = $IP_{DSL}$ |
| (M2) | $DS_L \rightarrow DS_Z$: | DNS_Query (**$ID_2$**, DN); dest. IP = $IP_{DSZ}$ |
| DNS cache poisoning (Until $ID_i = ID_2$) | | |
| (M3) | $ATT[DS_Z] \rightarrow DS_L$: | DNS_Reply ($ID_i$, DN, **$IP_{ATT}$**); dest. IP = $IP_{DSL} \times n$ |
| (M4) | $DS_L \rightarrow C$: | DNS_Reply ($ID_1$, DN, **$IP_{ATT}$**); dest. IP = $IP_C$ |
| All client HTTP traffic is sent to ATT directly | | |
| (M5) | $C \rightarrow ATT[WS]$: | HTTP_Request (URL); dest. IP = $IP_{ATT(WS)}$ |
| (M6) | $ATT[WS] \rightarrow C$: | HTTP_Response (HTML); dest. IP = $IP_C$ |

are used to authenticate website certificates, meaning that they cannot dispense SSL/TLS while browsing the Internet. In addition, transfer of the public key remains in plain text, meaning that it can still be forged using a MITM attack.

*4.2. Improvement or Enhancement on the Existing SSL/TLS.* Although SSL/TLS has been proposed for securing HTTP messages, as described in Section 2, attackers have discovered vulnerabilities [20]. The majority of attacks on SSL/TLS do not target the cryptographic core but instead exploit protocol vulnerabilities or intercept communicating nodes, as in MITM.

Typing the URL without HTTPS is a bad habit by users which exposes the communication to a stripping MITM. Numerous enforcement mechanisms have been proposed to prevent the success of MITM attacks, such as SHS-HTTPS [21], ISAN-HTTPS Enforcer [22], and HTTPSLock [23], all of which use client-side scripting to redirect the URL to HTTPS before sending the request. However, although this script enforces the request, it does not enforce the response, which may be from an attacker.

SSLock [24] and HSTS [25] use extra header fields which are attached by the web server, but few browsers or web servers support such method. The HTTPS enforcement technique is immune to stripping MITM attacks; however, protection from sniffing MITM attacks rests on the browser, which displays a warning message indicating a self-signed certificate. At this point, most users opt to accept by pressing the "safe" button, and this is another bad habit.

Aziz et al. [26] proposed extending TLS for integrity assurance against replay attacks and collusion attacks by using the *Trusted Platform Module* (TPM). TPM is a built-in hardware security chip embedded in the motherboard and is separate from the *central processing unit* (CPU). TPM includes cryptographic mechanisms for both host and program security. This approach is based on a hardware solution, which is not available to every user, and affords limited protection.

Elgohary et al. [27] have proposed an enhancement for SSL/TLS protocols by caching or storing a client session for future use, rather than repeating the entire communication process. However, the enhancement only benefits performance and not security.

## 5. The Proposed Scheme (Enc-DNS-HTTP)

The objective of this work is to secure Internet data transfer by securing web browsing or HTTP messages. The term "secure" refers to encryption and authentication which can withstand attacks.

In order to meet this objective, the proposed scheme distributes $PK_{WS}$ using a DNS server with BA authentication. The client and web server establish a session using $PK_{WS}$. At the beginning of the session, the two nodes negotiate the symmetric cipher technique and the key value to be used. The information is transferred securely, using the agreed authenticated key and techniques.

The proposed scheme does not change the Internet device infrastructure or the messaging procedure, and browsing continues to begin with DNS queries followed by HTTP messages. The proposed changes will be in the message contents, where we assume the following:

(i) BA has $PK_{BA}$, $RK_{BA}$, and $PK_{DS}$.

(ii) DNS server has $PK_{DS}$ and $RK_{DS}$; DNS server possesses $PK_{BA}$.

(iii) Every web server must have $DN_{WS}$, $IP_{WS}$, $PK_{WS}$, $RK_{WS}$, and $PK_{BA}$.

(iv) The client has $PK_{BA}$ through the browser and $PK_{DS}$ from the *Internet Service Provider* (ISP) during IP configuration.

Enc-DNS-HTTP consists of two phases, registration and Internet browsing, and these are described below.

*5.1. Registration Phase.* Web servers must be registered in a DNS server before they can be accessed through the Internet by clients. WS sends an encrypted request to BA, which contains $PK_{WS}$. Next, BA signs $PK_{WS}$ and sends an encrypted message to DS, where the term "sign" means to encrypt $PK_{WS}$ using $RK_{BA}$. The detailed registration protocol is described in Table 4. Then, WS information is stored in DS as $DN_{WS}$, whereas BA has signed both $IP_{WS}$ and $PK_{WS}$. The sequence of the protocol is shown in Figure 1.

*5.2. Internet Browsing Phase.* The Internet browsing phase refers to the client surfing the Internet. The protocol shown

Table 4: WS registration protocol.

| | | |
|---|---|---|
| | WS: | Buf = Enc ($PK_{WS}$, $DN_{WS}$, $IP_{WS}$; $PK_{BA}$) |
| (M1) | WS → BA: | Join_Request (Buf) |
| | BA: | Dec (Buf; $RK_{BA}$) = $PK_{WS}$, $DN_{WS}$, $IP_{WS}$ |
| | BA: | Buf = Enc ($DN_{WS}$, $IP_{WS}$; $PK_{DS}$) |
| (M2) | BA → DS: | Enquiry (Buf) |
| | DS: | Dec (Buf; $RK_{DS}$) = $DN_{WS}$, $IP_{WS}$ |
| | DS: | Search the database for $DN_{WS}$, $IP_{WS}$<br>If found then<br>Inq = Reject<br>Else<br>Inq = Accept<br>Buf = Enc (Inq; $PK_{BA}$) |
| (M3) | DS → BA: | Response_Enquiry (Buf) |
| | BA: | Dec (Buf; $RK_{BA}$) = Inq<br>If Inq == Reject then<br>Jr = Reject<br>Buf = Enc (Jr; $RK_{BA}$)<br>(M4) BA → WS: Join_Response (Buf)<br>Else<br>Jr= Accept<br>buf1 = Enc ($IP_{WS}$, $PK_{WS}$; $RK_{BA}$)<br>Buf = Enc ($DN_{WS}$, buf1; $PK_{DS}$) |
| (M4) | BA → DS: | Join (Buf) |
| | DS: | Dec (Buf; $RK_{DS}$) = $DN_{WS}$, buf1<br>Store the value in the database<br>Buf = success |
| (M5) | DS → BA: | Join_Reply (Buf) |
| (M6) | BA → WS: | Join_Response (Buf) |

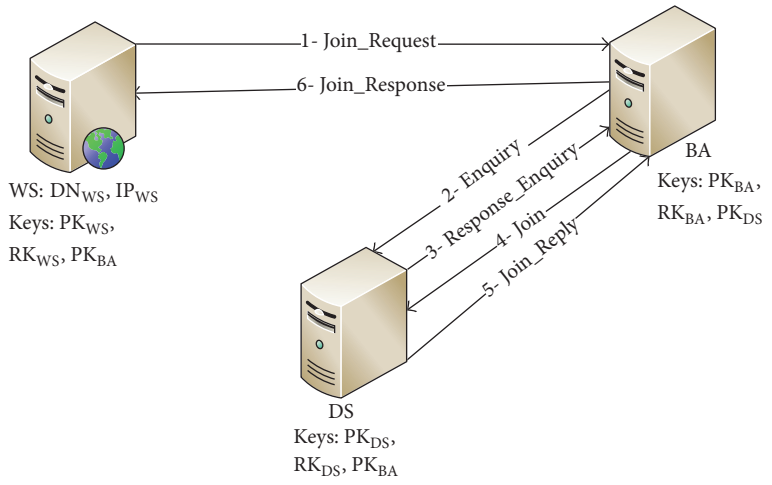


Figure 1: Registration phase.

in Table 5 presents the details of the Internet browsing phase, which begins by entering a URL in CB. CB then sends $DN_{WS}$ to the DNS resolver program. The resolver then extracts $IP_{DS}$ from the network setting, encrypts $DN_{WS}$ using $PK_{DS}$, and sends a query to DS. The reply carries $IP_{WS}$ and $PK_{WS}$ signed by BA, whereas the client obtains $IP_{WS}$ and $PK_{WS}$ by using $PK_{BA}$, which is stored in CB.

In more detail, the process is as follows: CBs generate $RN_C$ with CSL, encrypt them with $PK_{WS}$, and send the result as an HTTP message to WS using $IP_{WS}$. Next, WS generates $RN_{WS}$ encrypted with CSC and sends it to *C*. The hash function value from $RN_C$ and $RN_{WS}$ is SK for both sides. *C* uses $SK_C$ to encrypt the HTTP request message, and WS uses $SK_{WS}$ twice, in decrypting the client request and sending the encrypted

Table 5: WS Internet browsing protocol.

| | | |
|---|---|---|
| | $C$: | Open CB; user types a URL |
| | $C$: | CB extracts $DN_{WS}$ and delivers $DN_{WS}$ to the resolver |
| | $C$: | Buf = Enc ($DN_{WS}$; $PK_{DS}$) |
| (M1) | $C \rightarrow$ DS: | DNS_Query (Buf) |
| | DS: | Dec (Buf; $RK_{DS}$) = $DN_{WS}$<br>Search the database for $DN_{WS}$<br>If not found then<br>Buf = Not found<br>(M2) DS $\rightarrow$ $C$: DNS_Reply (Buf)<br>Else<br>Buf = Fetch values from DNS database |
| (M2) | DS $\rightarrow$ $C$: | DNS_Reply (Buf) |
| | $C$: | Dec (Buf; $PK_{BA}$) = $IP_{WS}$, $PK_{WS}$ |
| | $C$: | Generate $RN_C$ |
| | $C$: | Buf = Enc ($RN_C$, CSL; $PK_{WS}$) |
| (M3) | $C \rightarrow$ WS: | HTTP_RNC (Buf) |
| | WS: | Dec (Buf; $RK_{WS}$) = $RN_C$, CSL |
| | WS: | Generate $RN_{WS}$ |
| | WS: | $SK_{WS} = H(RN_C \mid\mid RN_{WS})$ |
| | WS: | Buf = Enc ($RN_{WS}$, CSC; $RK_{WS}$) |
| (M4) | WS $\rightarrow$ $C$: | HTTP_RNS (Buf) |
| | $C$: | Dec (Buf; $PK_{WS}$) = $RN_{WS}$, CSC |
| | $C$: | $SK_C = H(RN_C \mid\mid RN_{WS})$ |
| | $C$: | Buf = Enc (URL; $SK_C$) |
| (M5) | $C \rightarrow$ WS: | HTTP_Request (Buf) |
| | WS: | Dec (Buf; $SK_{WS}$) = URL |
| | WS: | Buf = Enc (info; $SK_{WS}$) |
| (M6) | WS $\rightarrow$ $C$: | HTTP_Response (Buf) |
| | $C$: | Dec (Buf; $SK_C$) = info |
| If there are no further messages, destroy session and delete $SK_C$ and $SK_{WS}$. | | |

response to $C$. If no further messages are transmitted between $C$ and WS, then both sides will delete SK. The sequence used by this protocol is illustrated in Figure 2.

## 6. Implementation

Ubuntu Linux 12.04 LTS [28] is used as a platform for implementation and experimentation. The proposed scheme is implemented with $C$ programming language, which allows network programming through the socket library. HTTP and DNS servers are implemented separately to be flexible and to manage DNS query and HTTP request programs implemented separately on the client side. For the purpose of ignoring user delays when typing URL, the client-side program reads URL from a file.

*6.1. Cryptography Programs.* RSA and SHA1 algorithms are implemented according to [29]. RSA key generation implemented stored each key in a different file to manage and distribute server keys. SHA1 result was stored in a file, which represented the symmetric cipher key.

The $C$ program code for triple DES was from OpenSSL [30] to make a fair comparison with SSL. *Cipher Block Chaining* (CBC) was utilised for triple DES operation mode. Triple DES uses three different 64-bit keys, which were provided by the keys derived from SHA1 result file.

The output of the cryptography algorithm is usually ambiguous unrecognized characters, which were compensated for by the implementation programs that read and transferred the results as hexadecimal numbers.

*6.2. DNS Programs.* DNS is divided into server-side and client-side programs. The server-side program listens on port 53 for incoming queries. When a client query arrives carrying $DN_{WS}$, the server replies with Enc($IP_{WS}$; $RK_{BA}$) in the answer section and Enc($PK_{WS}$; $RK_{BA}$) in the additional section of the message.

The client-side program sends a DNS query; this fetches $DN_{WS}$ from the URL file, creates a DNS query, and sends the query to the host, whose IP is saved in the *resolv.conf* file. Following this, the client-side program receives a DNS reply and extracts Enc($IP_{WS}$; $RK_{BA}$) and Enc($PK_{WS}$; $RK_{BA}$) from the server reply message. Finally, the

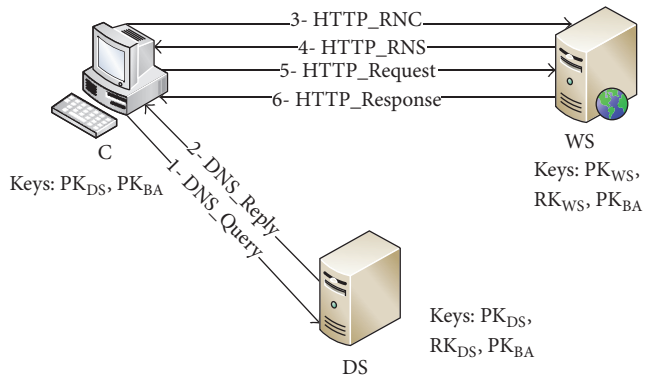


Figure 2: Internet browsing phase.

client-side program performs Dec(Enc($IP_{WS}$; $RK_{BA}$); $PK_{BA}$) and Dec(Enc($PK_{WS}$; $RK_{BA}$); $PK_{BA}$) to extract $IP_{WS}$ and $PK_{WS}$, which are delivered to the HTTP client program.

*6.3. HTTP Programs.* HTTP is programmed on both the server and client sides. The HTTP server listens on port 80 and has two response messages. The first message is the response to the client $RN_C$, which is $RN_{WS}$. These two values are protected using RSA, with Enc($RN_C$; $PK_{WS}$) and Enc($RN_{WS}$; $RK_{WS}$). When $RN_C$ and $RN_{WS}$ are entered in the SHA1 algorithm, the symmetric key file SK is produced on both sides. The second message contains the requested service, such as an HTML page, and this service is encrypted using the symmetric cipher agreed on in the previous message with SK.

The HTTP client-side program uses $IP_{WS}$, received from the DNS server, to set the destination address. The client program has two main messages. The first negotiates the symmetric cipher algorithm to be used with the server and shares the key parameter. The second message reads the URL from the URL file, encrypts the request using SK, and sends an HTTP request to the server. Finally, the client decrypts the received content from the server using SK and saves the result in a file.

## 7. Experimental Testing

A network with five hosts was built to test the Enc-DNS-HTTP protocol. Four hosts ran the Ubuntu Linux operating system, whereas one host ran Kali Linux [31], in order to use the attacker programs and tools. The first Ubuntu host represented WS; this ran the HTTP server-side program and possessed the cryptography programs. The second Ubuntu host represented DS and ran the DNS server-side program.

The third Ubuntu host operated as the client; this contained the DNS client program, the HTTP client program, and all cryptography programs. The client programs together made up CB. The final Ubuntu host had three network connections, in order to simulate the router, enable the *IP_forward* property for directing the packet to the correct network interface, connect all hosts to different IPs to simulate each host in a different network, enable DHCP-setting for the assignment of the DNS IP, and save IP in the *resolv.conf* file.

The client and the attacker were connected to the same LAN, as illustrated in Figure 3. Table 6 shows the host properties and contents. Prior to the experiment, all server keys were created using RSA key generation and stored in different files. It was assumed that the registration phase in the proposed scheme had been executed earlier, since this phase runs only once for each WS.

WS was installed with the Apache 2.4 HTTP server [32] and *C* was installed with the Curl 7 web browser [33] for the purpose of comparing results. WS was loaded with five HTML pages of different sizes: (a) HTML 100 Byte, (b) HTML 1 KB, (c) HTML 10 KB, (d) HTML 100 KB, and (e) HTML 1 MB. Each page was called four times in order to calculate the average time for the experiment. Wireshark software [34] was used for the capture and analysis of traffic.

In this experiment, the robustness of the proposed Enc-DNS-HTTP was tested under two conditions: firstly without an attack and then with an attack. The first condition represents the unsecured mode of Apache HTTP, the HTTP program, Apache HTTPS, and Enc-DNS-HTTP, whereas the second illustrates Apache HTTPS and Enc-DNS-HTTP in secure mode.

The experimental results from the proposed scheme in a real multisession are reported for the five different HTML pages. These pages were called four times under both conditions, and the performance of the proposed scheme was evaluated in terms of efficiency and effectiveness. The steps involved in the experimental procedure were as follows:

(i) Start the Wireshark program in WS, DS, and *C*.

(ii) Start the DNS and HTTP servers.

(iii) Use CB to call five pages, four times each, within 1 min.

(iv) Stop the DNS and HTTP servers.

(v) Stop the Wireshark program and save the packets captured in a file.

Table 6: Properties and contents of hosts.

| Host–Interface | | Properties | | Contents | |
|---|---|---|---|---|---|
| | | IP | MAC | Parameter files | Programs |
| WS – eth0 | | 192.168.33.2 | 00:0c:29:3a:fe:ce | $PK_{WS}$, $RK_{WS}$, $PK_{BA}$ | (i) HTTP server<br>(ii) Cryptography programs |
| DS – eth0 | | 192.168.22.2 | 00:0c:29:05:02:ac | $PK_{DS}$, $RK_{DS}$, $PK_{BA}$ | (i) DNS server<br>(ii) Cryptography programs |
| C – eth0 | | 192.168.11.20 | 00:0c:29:4d:6c:db | $PK_{BA}$, $PK_{DS}$ | (i) DNS client<br>(ii) HTTP client<br>(iii) Cryptography programs |
| ATT – eth0 | | 192.168.11.21 | 00:0c:29:c4:df:81 | $PK_{BA}$, $PK_{DS}$ | (i) ARP spoofing<br>(ii) DNS spoofing<br>(iii) SSL stripping |
| R | -eth0 | 192.168.11.1 | 00:0c:29:b2:00:05 | | |
| | -eth1 | 192.168.22.1 | 00:0c:29:b2:00:0f | | |
| | -eth2 | 192.168.33.1 | 00:0c:29:b2:00:19 | | |

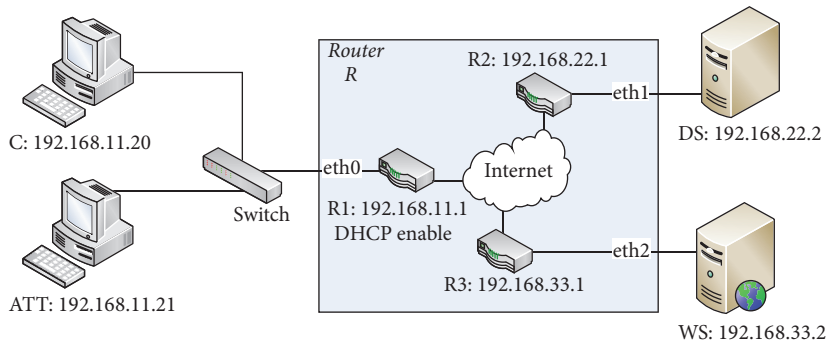


Figure 3: Experimental network.

This procedure was executed without attack for Apache HTTP, programmed HTTP, Apache HTTPS, and Enc-DNS-HTTP. In the attack situation run for HTTPS and Enc-DNS-HTTP, CB called each of the five pages once, and ATT ran the Wireshark program to capture network packets.

*7.1. Attacker Models.* The DNS spoofing attack procedure used was as follows:

(1) ATT connects to the LAN network and obtains an IP from *R*.

(2) ATT runs ARP poisoning to redirect packet transfers to his computer; ARP poisoning is performed by sending periodic ARP replies to *C* and *R*.

(3) ATT runs DNS spoofing to change $IP_{WS}$ in DNS reply, if $DN_{WS}$ in the packet matches DN in a file. ATT must know DN in advance.

(4) If $DN_{WS}$ matches, $IP_{WS}$ is changed to $IP_{ATT}$; in this experiment, this led to a DoS, as ATT did not implement a web page. Otherwise, the packet is forwarded to its destination.

The SSL stripping attack procedure used was as follows:

(1) ATT connects to the LAN network and obtains an IP from *R*.

(2) ATT runs ARP poisoning to redirect packet transfers to his computer; ARP poisoning is performed by sending periodic ARP replies to *C* and *R*.

(3) ATT runs SSL stripping to identify HTTPS traffic from *C* and forwards it to WS. When WS responds with HTTPS, ATT deceives *C* and responds with HTTP to *C*.

*C* obtains service through HTTP, rather than HTTPS, and ATT can read and modify all information from *C*.

## 8. Results and Discussion

The results indicate the robustness of Enc-DNS-HTTP; the scheme is implemented using the *C* programming language and focused on the essentials of Internet browsing. A comparison of this scheme with Apache HTTPS is inappropriate, since the Apache server is built on complex procedures which

require more time to execute. The results show that normal browsing with *C* programming language and the Apache HTTP server results in at least an approximate machine delay difference, which can be applied to encrypted browsing situations.

The results of this experiment show that Apache HTTP and programmed HTTP performed similarly in terms of operation time and media delay in unsecured conditions. In contrast, Enc-DNS-HTTP performs better than Apache HTTPS in the secure condition. In other words, Enc-DNS-HTTP can ensure the privacy of information transfer within the network.

In the discussion of the results given below, the term "normal HTTP" is used to refer to the programmed HTTP implemented without encryption and the normal HTTP Apache server. "Encrypted HTTP" refers to the programmed HTTP implemented with encryption as Enc-DNS-HTTP, as well as HTTPS run using the Apache server. "Encrypted HTTP under attack" refers to encrypted HTTP attacked by DNS spoofing.

*8.1. DNS Messages.* The messages between *C* and DS are exclusively DNS messages. Figure 4 shows the DS machine's average time delay, which is calculated from 20 DNS query–reply pairs, except for the attack situation, which is calculated from five DNS query–reply pairs. Figure 5 shows the media average time delay calculated for each DNS query–reply pair from the client side. The time difference between reply and query messages, minus the DS machine's time, is the media delay.

Normal HTTP shows minimal difference in Figure 4, due to the message sizes given in Table 7. A large variation in machine timing is observed in the encrypted HTTP, since Enc-DNS-HTTP uses cryptographic programs. Moreover, the message size is large, since $PK_{WS}$ is assigned to the additional section of the reply. However, since Apache HTTPS does not encrypt DNS messages, the value is approximately equal to that of normal HTTP.

Figure 4 clearly shows that the protocol proposed in this work requires more time, due to the use of the encryption process. The additional time cost of this scheme may be considered reasonable in order to achieve the security of DNS messages.

In the attack situation, no differences in terms of time were observed in the encrypted HTTP caused by the DS machine delay. However, media delay was affected, as shown in Figure 5, since DNS messages pass through the attacking machine, causing an additional delay. The DNS spoofing attack is based on a list of DNs stored in a file, and the DN is compared for each reply message. If one matches, the IP is then replaced; in this attack procedure, the media delay for HTTPS becomes large in the process of replacing $IP_{WS}$ value. It should be noted that the attacker cannot identify DN in the reply message when using the proposed scheme, since DN is encrypted with $PK_{DS}$, and only DS can produce the correct DN.

Under the proposed scheme, the DNS reply message is received by *C* carrying $IP_{WS}$, whereas, in Apache HTTPS, *C* carries the fake IP of the attacker. Although the DS machine time is increased with the use of Enc-DNS-HTTP, this scheme protects the browser from attack.

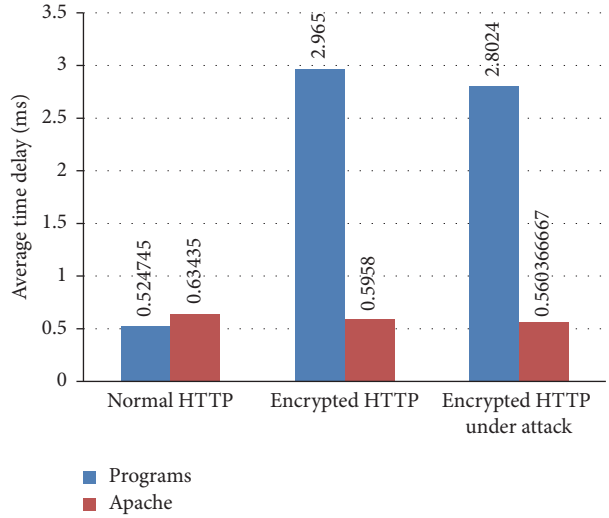


Figure 4: DS machine average time delay.

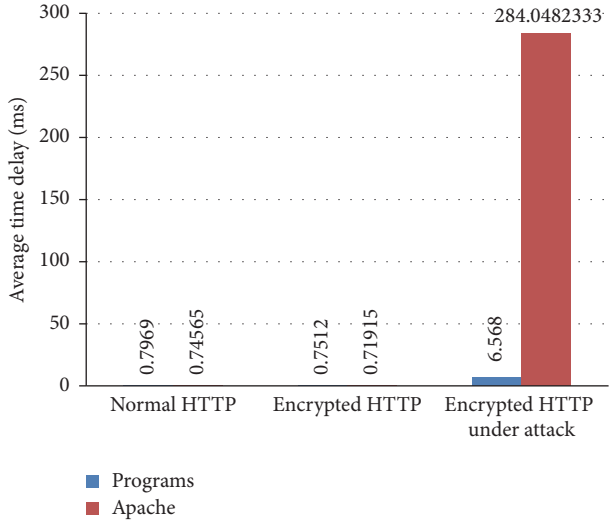


Figure 5: Average media time delay between *C* and DS.

Again, as can be seen in Figure 5, the Apache HTTPS induces a higher delay, since the time taken for the ATT to find the $DN_{WS}$ match in the DNS response and subsequently replace $IP_{WS}$ with $IP_{ATT}$ causes a delay in the response.

*8.2. TCP Messages.* *C* and WS communicate through TCP messages, which carry user requests in the form of URLs. WS responds through HTML. The average time delays for the WS machine, shown in Figure 6, demonstrate that Enc-DNS-HTTP is superior even under attack. No curve is shown for Apache HTTPS in an attack situation, since this is vulnerable to the attack, causing a DoS for *C*.

Figure 6 indicates that the page size affects the WS delay. After the fourth page, machine delay scatters, and Apache requires the largest time for both HTTP and HTTPS.

Table 7: Sizes of DNS messages.

| | Programmed HTTP | Apache HTTP | Programmed Enc-DNS-HTTP | Apache HTTPS |
|---|---|---|---|---|
| Query message size (bytes) | 68 | 75 | 91 | 75 |
| Reply message size (bytes) | 84 | 91 | 125 | 91 |

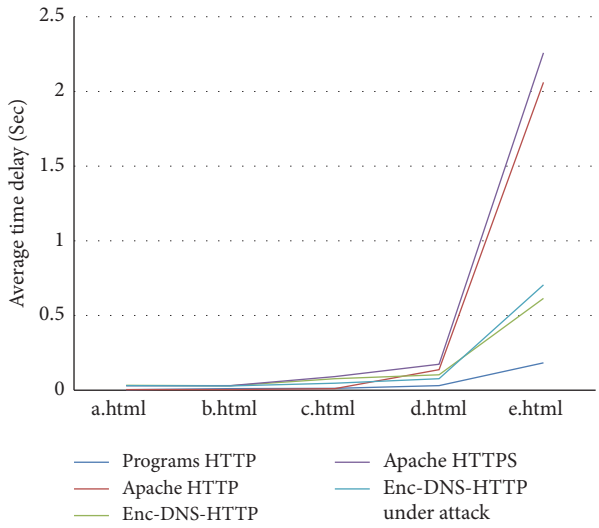


Figure 6: Average time delay for WS machine.

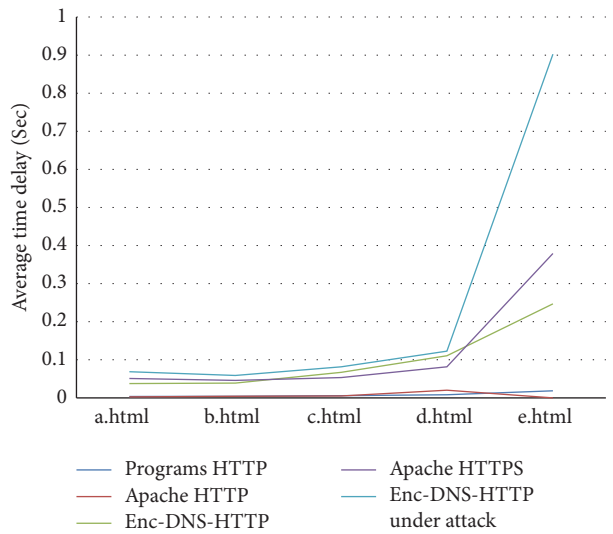


Figure 7: Average time delay for the media between *C* and WS.

The delay time of Enc-DNS-HTTP under attack is due solely to the use of ARP poisoning by the attacker before DNS spoofing; this sends ARP replies to WS, creating additional tasks for WS. The curve for Enc-DNS-HTTP in Figure 6 clearly shows that the process of an attack has no effect on our scheme.

Figure 7 shows increases in media delay when Enc-DNS-HTTP is under attack; in this case, packets pass through the attacker's machine, adding an extra delay. The attacker forwards only TCP messages, as he cannot identify DN, which is encrypted. The performance of both the Apache HTTPS and our scheme within the framework at transmission time is evaluated, as shown in Figure 7. The differences in time delay illustrate that the proposed scheme is faster than Apache HTTPS; this is due to the lower number of negotiation messages required by the proposed scheme.

HTTPS shows a higher media delay than Enc-DNS-HTTP, due to the number of messages transferred within the Handshake protocol. Enc-DNS-HTTP requires only two messages, whereas HTTPS requires eight, as discussed in Section 2.2.2. The number of handshake messages affects the WS machine time; this can be seen in Figure 8, where HTTPS has the largest value. Figure 8 also demonstrates the correctness of the proposed scheme in terms of server load. As expected, our scheme has a lower impact on machine delay with increasing message size, since it employs few time-consuming operations, unlike Apache HTTPS.

Thus, Apache HTTPS and Enc-DNS-HTTP are tested on the same types of pages carried by TCP. These tests show that the number and size of negotiation messages have

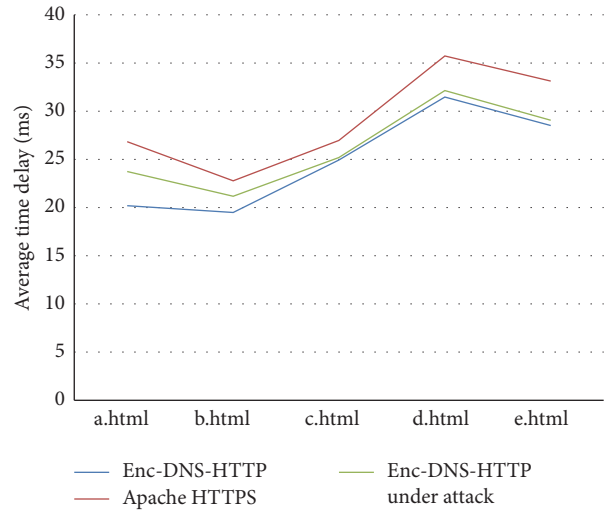


Figure 8: Average time delay for WS machine (handshake only).

a large influence over the results, leading to Enc-DNS-HTTP outperforming Apache HTTPS.

*8.3. Throughput.* The throughput of both our scheme and Apache is evaluated in three cases: normal, encrypted, and encrypted under attack. As can be seen in Figure 9, in the first case both methods are comparable, with average throughput; however, in the second case the difference is clear considering the DNS message size. The third case illustrates that our

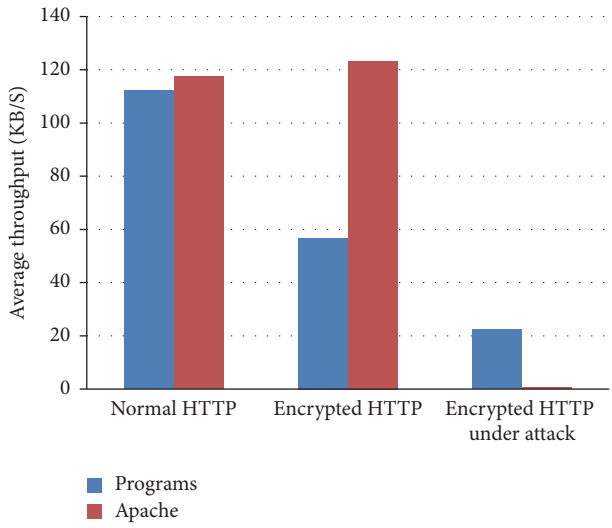


Figure 9: Throughput between $C$ and DS.

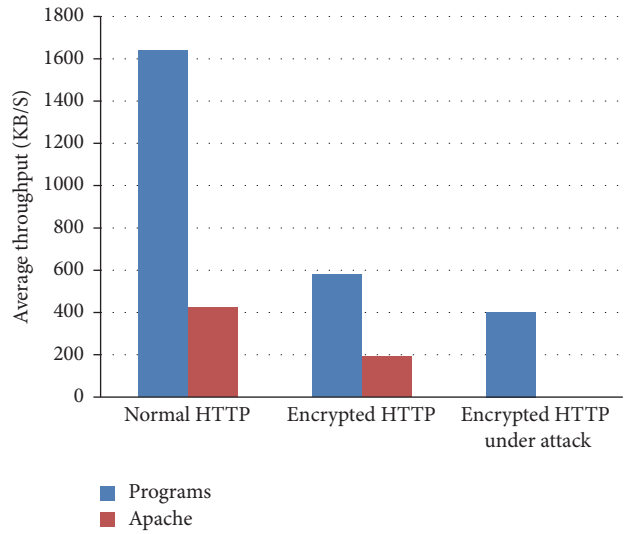


Figure 10: System throughput.

work is free from breach by attack and that it induces better throughput than Apache.

The difference in throughput between $C$ and DS shown in Figure 9 is due to the size of the message. Enc-DNS-HTTP has larger DNS messages than HTTPS, resulting in a reduced throughput. Attacker interception reduces throughput, increasing the packet time delay; since the throughput of Enc-DNS-HTTP is higher than that of Apache, this is further confirmation that the attacker did not modify the reply, unlike in the case of HTTPS.

Combining the DNS and TCP message sizes and considering the request–response delay from $C$ side gives system throughput, as presented in Figure 10. Securing HTTP will reduce system performance, which leads to decreased throughput. Although DNS spoofing lowers throughput, the attacker cannot break browsing security with fake IP in a DNS reply. However, when Apache HTTPS is used, the attacker can manipulate the DNS replies with a fake IP, resulting in no HTTPS response; this gives an infinite delay and a throughput equal to zero.

In summary, the entire protocol is analysed here in order to test its performance in terms of average throughput; the proposed scheme clearly has better throughput than Apache. In addition, the security ability of the proposed scheme is tested; our scheme performed well under the well-known Apache HTTPS attacks, since this scheme secures DNS messages and shares the WS public key, unlike Apache.

Enc-DNS-HTTP withstands DNS spoofing and SSL stripping attacks, which read the HTTPS request–response process between $C$ and WS. With Enc-DNS-HTTP, the attacker cannot deceive $C$ and redirect to HTTP, since the first request is encrypted.

## 9. Attack Definitions and Security Analysis

In this section, an illustration of the most well-known attacks that threaten web browsing security is presented. Following this, the security of the proposed scheme will be described and analysed.

*9.1. Attacks.* The theft of sensitive data from users is one of the most frequent objectives pursued by attackers. Numerous methods are employed to tempt users into providing their data over the wrong connection, which leads to the attacker's destination rather than the legitimate destination.

HTTPS offers one defence against web attacks, but this suffers from vulnerabilities [35]. However, the goal of attackers is generally to impersonate the server, rather than to crack HTTPS keys. The attacker intercepts traffic from the source and forwards it to the destination (and vice versa), creating an illusion of the user and server being connected normally, when in fact the attacker can modify messages and insert new ones. The most well-known attacks are detailed in the following subsections [36, 37].

*9.1.1. MITM.* According to RFC 2828, a MITM attack is "a form of active wiretapping attack in which the attacker intercepts and selectively modifies communicated data in order to masquerade as one or more of the entities involved in a communication association." In a virtual sense, the attacking machine is placed between two communicating computers, giving the attacker the capability to read, modify, or block information. The original computers believe they are connected only to each other, and neither the user nor the server is aware of the MITM [38, 39].

According to [22, 40] the MITM attack may employ two methods: "stripping" and "sniffing." In MITM sniffing, a forged self-signed certificate is presented to the victim as a legal web server certificate. After the user accepts the fake certificate, the user's information is compromised. However, a MITM stripping attack changes an HTTPS connection with the victim to HTTP and connects with the web server using HTTPS. The use of the HTTP connection causes the victim's information to be transferred in plain text format.

*9.1.2. Denial of Service (DoS).* The growing number of DoS attacks imposes a significant threat to the availability of network services. A DoS attack is characterised by malicious behaviour, which prevents legitimate users of a network service from using that service [41, 42]. There are two principal types of DoS attack. The first is a flooding attack, which sends an excessive quantity of packets to a client victim. These packets arrive in such high quantity that a certain key resource at the victim (bandwidth, buffers, or CPU time to compute responses) is quickly depleted. The victim machine either crashes or spends so long compensating for attack traffic that it cannot attend to its real work [43]. The second type of DoS attack starts as a MITM attack; however, when the attack blocks the packets sent to the client, the attack becomes a DoS.

*9.2. Security Analysis*

**Theorem 1.** *Enc-DNS-HTTP is protected from DNS spoofing attack.*

*Proof.* As mentioned in Section 7.1, in order to accomplish DNS spoofing, the attacker must know $DN_{WS}$ in advance and store this in a file. Enc-DNS-HTTP sends encrypted $DN_{WS}$ using $PK_{DS}$ in DNS query, and WS replies with the same encrypted $DN_{WS}$, as shown in the (M1) and (M2) messages in Table 5. The encryption of $DN_{WS}$ means that $DN_{WS}$ will not match the DN known by ATT. □

**Theorem 2.** *If the DNS spoofing attack stores the encrypted* $DN_{WS}$*, then Enc-DNS-HTTP will not disclose C information.*

*Proof.* ATT may have $PK_{DS}$, since ATT is on the same LAN as *C*. If ATT encrypts a list of DN and stores the ciphered DN in a DNS spoofing file, then the website will be identified from the *C* query, since both *C* and ATT possess $PK_{DS}$. ATT saves the DNS query $ID_C$ and compares the ID of all DNS replies with $ID_C$. Once ATT finds the matching/desired website, ATT can change $IP_{WS}$.

*C* will then suffer from DoS, since *C* will decrypt $IP_{WS}$ from the DNS reply, as shown in message (M2) in Table 5, leading to a random IP. The HTTP request will receive no response, leading to DoS. ATT can achieve DoS but cannot read the information of *C*. □

**Theorem 3.** *A DNS spoofing attack is unable to replace* $IP_{WS}$ *with a legal IP.*

*Proof.* ATT can read both the DNS reply and $IP_{WS}$ by decrypting with $PK_{BA}$, which accompanies CB. However, ATT cannot replace $IP_{WS}$ with a real IP, since ATT does not have $RK_{BA}$ to reencrypt the desired $IP_{ATT}$. □

**Theorem 4.** *Enc-DNS-HTTP is protected from DNS poisoning attack.*

*Proof.* Table 3 shows that DNS cache poisoning attacks depend on guessing $ID_2$ when $IP_{WS}$ is stored in a regular format. However, in Enc-DNS-HTTP, $IP_{WS}$ is encrypted with BA authentication using $RK_{BA}$ when saved, as illustrated by message (M3) in Table 4. If ATT is lucky and guesses $ID_2$, the stored record will be corrupted because $PK_{WS}$ is an empty field; thus, the IP does not lead to WS for *C*. □

**Theorem 5.** *Enc-DNS-HTTP prevents a MITM attack from disclosing C information.*

*Proof.* As described in Section 7.1, the first step for an ATT is to execute ARP poisoning in order to reside in a virtual sense between *C* and *R*; this results in both *C* and *R* sending all network packets through ATT. From the *C* standpoint, ATT masquerades as *R*; from the *R* standpoint, ATT masquerades as *C*. However, even if all packets pass through ATT, ATT cannot read or forge any information, since the information is encrypted using an asymmetric cipher. *C* encrypts data using $PK_{DS}$ for sending to DS and encrypts data using $PK_{WS}$ for sending to WS. In both steps, ATT is unable to read the data, as ATT has neither $RK_{DS}$ nor $RK_{WS}$. *C* and WS agree on a symmetric algorithm and this is the key to guaranteeing the protection of the information. □

**Theorem 6.** *Enc-DNS-HTTP is protected from a MITM stripping attack.*

*Proof.* As described in Section 7.1, in MITM stripping ATT will identify HTTPS traffic and change it to HTTP. ATT will forward a packet from *C*, receive the WS response, decrypt the response, and send an HTTP response to *C*. In Enc-DNS-HTTP, ATT cannot perform the second stage because he does not have $PK_{WS}$, which is obtained by *C* from DS. □

**Theorem 7.** *Enc-DNS-HTTP is protected from a MITM sniffing attack.*

*Proof.* As described in Section 9.1.1, MITM sniffing attacks circumvent HTTPS by presenting a forged certificate, after which ATT waits for acceptance by *C*. With Enc-DNS-HTTP, ATT cannot forge $PK_{WS}$ without $RK_{BA}$, and ATT must poison the DNS cache. If ATT desires to fake $IP_{WS}$, then this becomes a redirection task, as demonstrated in Theorems 1 and 2. □

**Theorem 8.** *Enc-DNS-HTTP supports WS authentication.*

*Proof.* As described in Section 5.2, the handshake information transferred from WS is encrypted using $RK_{WS}$, which is possessed only by WS. Following this, the information from WS is authenticated. □

**Theorem 9.** *Enc-DNS-HTTP supports data transfer security.*

*Proof.* As described in Section 5.2, data are encrypted before being transferred across the network. Both asymmetric and symmetric encryption are used for the data; the symmetric key is discarded after each session, a form of one-time password. □

## 10. Conclusion

The security of web browsing depends on SSL/TLS to secure a client and web server connection. MITM and

DNS attacks threaten the privacy of HTTPS using different approaches.

This paper proposes Enc-DNS-HTTP to protect web browsing and to secure client–DNS server and client–web server communications. The scheme is based on sharing a web server public key through the DNS server. The key is signed by a trusted third party, such as a web browser program creator.

The browser program begins by fetching a web server public key from a DNS server and verifying the key through a third party public key ($PK_{BA}$). Following this, the browser program sends an encrypted symmetric key parameter to the web server. After receiving the second symmetric key parameter from the server, both sides generate a secret key. Finally, the browser program requests the service from the web server. The request is encrypted with the secret key and the response will be encrypted with the same key.

The proposed scheme is implemented in the C programming language on a Linux platform. The results demonstrate the effectiveness of Enc-DNS-HTTP in protecting web browsing. In addition, throughput shows improved performance, despite the encryption affecting the communication from both the DNS and web servers.

## Notations

*Parameter*

| | |
|---|---|
| PK: | Public key |
| IP: | Internet protocol |
| RN: | Random number |
| Info: | Information or HTML page |
| CSL: | Cipher suite list supported by the client node |
| CSC: | Cipher suite choice |
| CB: | Client browser program |
| RK: | Private key |
| URL: | Uniform resource locator |
| SK: | Secret key/session key |
| S_ID: | Session ID |
| Msg_all: | All messages exchanged between $C$ and WS so far |
| Sig: | Authority signature |
| DN: | Domain name. |

*Function*

| | |
|---|---|
| $\mathrm{Enc}(X; Y)$: | $X$ is encrypted with key $Y$ |
| $\mathrm{Dec}(X; Z)$: | $X$ is decrypted with key $Z$ |
| $\mathrm{Enc}(X, W, Z; Y)$: | Each parameter encrypted separately with $Y$ |
| $H(X)$: | Hashed value of $X$ |
| $H(X \parallel Y)$: | Hashed value after concatenating $X$ and $Y$. |

## Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

## Acknowledgments

This work is supported by National 973 Fundamental Basic Research Program of China under Grant no. 2014CB340600.

## References

[1] Z. Ye, S. Smith, and D. Anthony, "Trusted paths for browsers," *ACM Transactions on Information and System Security*, vol. 8, no. 2, pp. 153–186, 2005.

[2] A. Herzberg and A. Jbara, "Security and identification indicators for browsers against spoofing and phishing attacks," *ACM Transactions on Internet Technology (TOIT)*, vol. 8, no. 4, pp. 16:1–16:36, 2008.

[3] B. A. Forouzan, *TCP/IP Protocol Suite*, McGraw-Hill, 4th edition, 2010.

[4] J. Du and G. Nie, "Design and implementation of security reverse data proxy server based on SSL," in *Proceedings of the Proceedings of International Conference on Communications in Computer and Information Science (ICCIC '11)*, pp. 523–528, Wuhan, China, 2011.

[5] K. Bhargavan, C. Fournet, R. Corin, and E. Zălinescu, "Verified cryptographic implementations for TLS," *ACM Transactions on Information and System Security*, vol. 15, no. 1, article no. 3, 2012.

[6] A. Bates, J. Pletcher, T. Nichols, B. Hollembaek, D. Tian, and K. R. B. Butler, "Securing SSL certificate verification through dynamic linking," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS '14)*, pp. 394–405, ACM, Scottsdale, Ariz, USA, November 2014.

[7] H. Lee, T. Malkin, and E. Nahum, "Cryptographic strength of SSL/TLS servers: current and recent practices," in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement (IMC '07)*, pp. 83–92, ACM, San Diego, USA, Calif, USA, 2007.

[8] C. Castelluccia, E. Mykletun, and G. Tsudik, "Improving secure server performance by Re-balancing SSL/TLS handshakes," in *Proceedings of the ACM Symposium on Information, Computer and Communications Security (ASIACCS '06)*, pp. 26–34, IEEE, Taipei, Taiwan, March 2006.

[9] J. GroBschadl and I. Kizhvatov, "Performance and security aspects of client-side SSL/TLS processing on mobile devices," in *Proceedings of the 9th International Conference on Cryptology and Network Security (CANS '10)*, pp. 44–61, Springer, Kuala Lumpur, Malaysia, December 2010.

[10] T. Saito, K. Sekiguchi, and R. Hatsugai, "Authentication binding between TLS and HTTP," in *Proceedings of the 2nd International Conference on Network-Based Information Systems (NBiS '08)*, pp. 252–262, Springer, Turin, Italy, September 2008.

[11] H. Yang, E. Osterweil, D. Massey, S. Lu, and L. Zhang, "Deploying cryptography in internet-scale systems: a case study on DNSSEC," *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 5, pp. 656–669, 2011.

[12] C. Shue and A. Kalafut, "Resolvers revealed: characterizing DNS resolvers and their clients," *ACM Transactions on Internet Technology (TOIT)*, vol. 12, no. 4, pp. 14:1–14:17, 2013.

[13] R. van Rijswijk-Deij, A. Sperotto, and A. Pras, "DNSSEC and its potential for DDoS attacks: a comprehensive measurement study," in *Proceedings of the ACM Internet Measurement Conference (IMC '14)*, pp. 449–460, ACM, Vancouver, Canada, November 2014.

[14] H. Wu, X. Dang, L. Zhang, and L. Wang, "Kalman filter based DNS cache poisoning attack detection," in *Proceedings of the*

*11th IEEE International Conference on Automation Science and Engineering (CASE '15)*, pp. 1594–1600, August 2015.

[15] D. Gollmann, "Secure applications without secure infrastructures," in *Proceedings of the 5th International Conference on Mathematical Methods, Models and Architectures for Computer Network Security (MMM-ACNS '10)*, pp. 21–31, Petersburg, Russia, 2010.

[16] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, "DNS security introduction and requirements," RFC Editor RFC4033, 2005.

[17] A. Herzberg, H. Shulman, and B. Crispo, "Less is more: cipher-suite negotiation for DNSSEC," in *Proceedings of the 30th Annual Computer Security Applications Conference (ACSAC '14)*, pp. 346–355, ACM, New Orleans, La, USa, December 2014.

[18] P. Hoffman and J. Schlyter, "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA," Internet Engineering Task Force, RFC 6698, 2012.

[19] P. Hallam-Baker and R. Stradling, *DNS Certification Authority Authorization (CAA) Resource Record*, Internet Engineering Task Force, RFC 6844, 2013.

[20] O. Levillain, A. Ebalard, B. Morin, and H. Debar, "One year of SSL internet measurement," in *Proceedings of the 28th Annual Computer Security Applications Conference (ACSAC '12)*, pp. 11–20, ACM, Orlando, Fla, USA, December 2012.

[21] B. Sugavanesh, R. Hari Prasath, and S. Selvakumar, "SHS-HTTPS enforcer: enforcing HTTPS and preventing MITM attacks," *ACM SIGSOFT Software Engineering Notes*, vol. 38, no. 6, pp. 1–4, 2013.

[22] S. Puangpronpitag and N. Sriwiboon, "Simple and lightweight HTTPS enforcement to protect against SSL striping attack," in *Proceedings of the 4th International Conference on Computational Intelligence, Communication Systems and Networks (CICSyN '12)*, pp. 229–234, Phuket, Thailand, July 2012.

[23] A. P. H. Fung and K. W. Cheung, "HTTPSLock: enforcing HTTPS in unmodified browsers with cached Javascript," in *Proceedings of the 4th International Conference on Network and System Security (NSS '10)*, pp. 269–274, IEEE, Melbourne, Australia, September 2010.

[24] A. P. H. Fung and K. W. Cheung, "SSLock: sustaining the trust on entities brought by SSL," in *Proceedings of the 5th ACM Symposium on Information, Computer and Communication Security (ASIACCS '10)*, pp. 204–213, ACM, Beijing, China, April 2010.

[25] J. Hodges, C. Jackson, and A. Barth, *HTTP Strict Transport Security (HSTS)*, Internet Engineering Task Force, RFC 6797, 2012.

[26] N. Aziz, N. Udzir, and R. Mahmod, "Extending TLS with mutual attestation for platform integrity assurance," *Journal of Communications*, vol. 9, no. 1, pp. 63–72, 2014.

[27] A. Elgohary, T. S. Sobh, and M. Zaki, "Design of an enhancement for SSL/TLS protocols," *Computers and Security*, vol. 25, no. 4, pp. 297–306, 2006.

[28] Linux Ubuntu, http://www.ubuntu.com/.

[29] W. Stallings, *Cryptography and Network Security: Principles and Practice*, Prentice Hall, 5th edition, 2011.

[30] OpenSSL, https://www.openssl.org/.

[31] Kali Linux, https://www.kali.org/.

[32] Apache Web Server, https://httpd.apache.org/.

[33] Curl https://curl.haxx.se/.

[34] Wireshark, https://www.wireshark.org/.

[35] A. Eldewahi, T. Sharfi, A. Mansor, N. Mohamed, and S. Alwahbani, "SSL/TLS attacks: analysis and evaluation," in *Proceedings of the International Conference on Computing, Control, Networking, Electronics and Embedded Systems Engineering (ICCNEEE '15)*, pp. 203–208, IEEE, Khartoum, Sudan, 2015.

[36] Y. Jia, Y. Chen, X. Dong, P. Saxena, J. Mao, and Z. Liang, "Man-in-the-browser-cache: persisting HTTPS attacks via browser cache poisoning," *Computers and Security*, vol. 55, no. 1, pp. 62–80, 2015.

[37] M. Prandini and M. Ramilli, "A browser-based distributed system for the detection of HTTPS stripping attacks against web pages," in *Proceedings of the 27th IFIP TC 11 Conference on Information Security and Privacy (SEC '12)*, pp. 549–554, Springer, Heraklion, Greece, June 2012.

[38] J. Du, X. Li, and H. Huang, "A study of man-in-the-middle attack based on SSL certificate interaction," in *Proceedings of the 1st International Conference on Instrumentation, Measurement, Computer, Communication and Control (IMCCC '11)*, pp. 445–448, IEEE, Beijing, China, October 2011.

[39] D. Berbecaru and A. Lioy, "On the robustness of applications based on the SSL and TLS security protocols," in *Proceedings of the 4th European PKI Workshop on Public Key Infrastructure (EuroPKI '07)*, pp. 248–264, Springer, Palma de Mallorca, Spain, 2007.

[40] K. Cheng, M. Gao, and R. Guo, "Analysis and research on HTTPS hijacking attacks," in *Proceedings of the 2nd International Conference on Networks Security Wireless Communications and Trusted Computing (NSWCTC '10)*, pp. 223–226, IEEE, Wuhan, China, April 2010.

[41] M. S. Fallah, "A puzzle-based defense strategy against flooding attacks using game theory," *IEEE Transactions on Dependable and Secure Computing*, vol. 7, no. 1, pp. 5–19, 2010.

[42] H. Wang, D. Zhang, and K. G. Shin, "Change-point monitoring for the detection of DoS attacks," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 4, pp. 193–208, 2004.

[43] J. Mirkovic and P. Reiher, "D-WARD: a source-end defense against flooding denial-of-service attacks," *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no. 3, pp. 216–232, 2005.