PERIODICA POLYTECHNICA SER. EL. ENG. VOL. 47, NO. 1–2, PP. 71–88 (2003)

OVERHEAD ANALYSIS OF HTTPS

László ZÖMBIK

Department of Telecommunication and Telematics
Budapest University of Technology and Economics
H–1117, Magyar tudósok körútja 2, Budapest, Hungary
e-mail: laszlo.zombik@ericsson.com

Received: Feb. 2, 2003; Revised: July 22, 2003

Abstract

Secure web access has a remarkable growth. Users would like to exploit the advantages of the Internet for online banking, for e-commerce or they simply would like to protect their information e.g. with using a secure web mailer. Https is a simple http traffic on top of a security protocol (e.g. SSL, TLS) is used for serving this need. This paper gives detailed analysis of https traffic to aid traffic dimensioning or traffic modelling and even to assist investigation of traffic flow confidentiality.

Keywords: https, SSL, TLS, traffic analysis, traffic flow confidentiality.

1. Introduction

The HTTP [5] has the largest traffic on the Internet. Users primarily surf on the Internet to collect public information. Hovewer, the number of the web-based commercial services (e-commerce) or private accesses like e-banking rapidly grows. In addition, protection of letters using web-based e-mail services has also a vital necessity. These services require to protect communication between peers, moreover between users.

Without protecting sensitive web traffic, an eavesdropper may deduce parameters or user behaviour, or an attacker can even identify and impersonate the victim. Therefore, several security solutions have evolved, the two most notable security protocols are the SSL [1] and the TLS [3]. The two protocols are very similar. Before sending user information, the client and the server initially negotiate about the security association, including cryptographic algorithms and keys. After the handshake phase, protected traffic is transmitted. If a web server uses SSL or TLS, then the HTTP [4, 5] communication is secured. In this case the communication is called HTTPS[11].

It is beneficial to have a model for https for bandwidth or for cost estimation. This is extremely important for communications that contain expensive links or the link capacity is limited and secure web access is required. An example can be when the subscribers use satellite terminals, GPRS or UMTS based equipment for browsing on the web.

This paper introduces a methodology for modelling traffic characteristics of security protocols in Section 2. Based on this methodology, a traffic model for

https is presented. In Section 3 security, protocols of https are shown, Section 4 introduces a generic https model. This model is used for the characterisation of https traffic in Section 5. Section 6 concerns the user behaviour and the properties of transport and lower layers which can affect the shape of the https traffic. The traffic model is verified by measurements in Section 7.

2. Methodology for Modelling Traffic Characteristics of Security Protocols

Traffic characteristics of any communication protocol can be modelled easily, since the message format, length and behaviour are known from protocol specifications. However, time-dependence and options (ambiguously specified behaviour) in the specification can cause statistical properties of traffic characteristics. This kind of statistical properties still can be characterised by traffic measurements. Security protocols, unfortunately, lose the property of observability due to employment of data confidentiality, so from their traffic measurements one can hardly deduce statistical properties. Even most of the security protocols possess the ability of traffic flow confidentiality. This not only hides the actual value of data elements, but also the length and the place of information is concealed. In test envinronment, all information (e.g. secret material) is available for characterising a security protocol, however, its behaviour can be quite different than the real-time traffic. Furthermore, applied methodology in test envinronments for characterisation of real-time traffic is not manageable. Therefore it is impossible to find an accountable network administrator (e.g. of an on-line bank) who agrees to disclose confidential information for characterisation.

The following methodology tries to address this issue, it is recommended for investigation of real-time traffic without the need of any confidential information. It requires to know the behaviour of the security protocol, like message flow or where and how cryptographic algorithms, random padding lengths, or compression methods are used.

Step 1: message attributes or variables (like port number, length field), used in all types of implementations, should be identified.

Step 2: explore which cryptographic and compression algorithms might occur in the security protocol.

Step 3: determine or estimate in what way a specific algorithm affects the message attributes or even the whole message (e.g. an MD5 [6] hash increases the length of message by 16 bytes).

Step 4: determine the value of those attributes which do not depend on the various cryptographic and compression algorithms, but have either fixed length (e.g. Version) or have variable length (e.g. SPI). From the protocol specification or measurements, the probability density function of the attributes with variable length can be determined or estimated.

Step 5: For those parameters, which depend on some cryptographic or compression algorithms, determine the statistical properties (e.g. the probability of

occurrence) of a specific algorithm per message attributes. From the protocol specification or measurements (by tracing the list of offered and accepted ciphersuites) this probability can be determined.

Step 6: with characterisation of upper and lower layer protocols, the effect to the security protocol can be determined. For example, an influence from upper layer is the user behaviour. From the lower layer propagation, buffering delays, and other network effects originate. Existing traffic models like HTTP, TCP source models can be used here.

Step 7: Model statistical properties of communication, like interarrival time between messages, correlation, etc.

Step 8: Simplify the model by neglecting infrequent messages or rare ciphersuites, or approximating complex messages, and those lengths.

Step 9: Evaluate the model to measurements and feed back the results.

Chapter 4 addresses the first step, while the next four steps and Step 8 are covered in chapter 5. Step 6 is discussed in chapter 6, and the results are evaluated in chapter 7.

3. Security Protocols for https

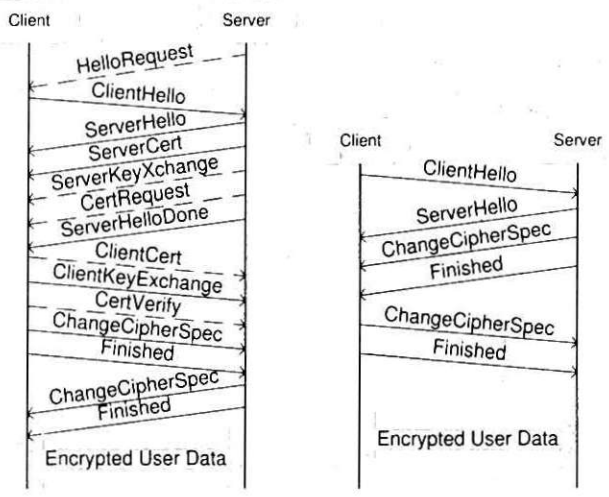
Https can choose between three security protocols. However, the oldest one is the Secure Socket Layer version 2 (SSLv2) [2] which contains several security flaws, therefore this is extremely seldom used. Corrected version of this protocol, the Secure Socket Layer version 3.0 rapidly became de facto standard. Later the Internet Engineering Task Force (IETF) released an official standard, based on the SSLv3.0, which is called Transport Layer Security (TLS). The difference between the TLS and the SSLv3.0 is minimal: the number of ciphersuites, alert, and certificate types were extended, the method of key calculation was modified and the Handshake finalisation message was modified (simplified). The version numbering of TLS shows (version 3.1) that this protocol can be considered as the successor of SSLv3.0. Nowadays, secure http communication prefers SSLv3.0, but the latest versions of applications start to use TLS only.

Fig. 1 shows the SSL/TLS message flow. Messages with dashed lines are optional. *Fig. 1(a)* shows the full handshake message exchange. The SSL/TLS communication starts with the ClientHello messages, where the client sends several ciphersuites to initiate a session. The server selects one ciphersuite and answers with a ServerHello message; also certificate of the server and, optionally, if the selected ciphersuite requires, serverKeyExchange message is attached. If the server requires client certificate for authentication, then this is indicated in the ClientCertRequest message. Finally, the ServerHelloDone closes this communication. If a client certificate is requested, then the client attaches its certificate in the answer, also responds with ClientKeyExchange, which contains the keying material. Certificate Verify notification may also be sent by the client. After negotiating and sending the keying material, secure communication should be started. The ChangeCipherSpec

message indicates, that the negotiated security procedures must be used right after this message. Therefore, the last message of the client, the Finished is encrypted. If the server receives the ClientKeyExchange, it starts to calculate secret keys. If this is finished, the server answers to the client with a ChangeCipherSpec message. The encrypted Finish message of the server closes the handshake. After the handshake phase, peers exchange encrypted user data.

If both the server and the client support the Session Key Caching (SKC) mechanisms, then after the initial communication they can improve efficiency by reusing existing information. The reduced message flow is presented in *Fig. 1(b)*.

In the SSL/TLS protocol stack upper layer messages are encapsulated and protected in the lower SSL/TLS Record Layer. There are different methods to achieve confidentiality. *Stream ciphers* have the same length as the plaintext; *block ciphers* may have padding in order to fill the plaintext to block size of the ciphering algorithm. The padding can be up to 255 bytes in order to provide some data flow confidentiality too. Record Layer messages, furthermore, contain a generic header (which consists of Type, Version and Length fields) and a Message Authentication Code (MAC).



(a) Full Handshake (b) Session Key Caching

Fig. 1. The SSL/TLS message flow

4. General https Model

We denote the user traffic by $f(t)$, let θ ($\theta : \Omega \mapsto \Re$) model the interarrival time of messages $f(t) = \text{msg}(t)I(\theta < t)$. I(.) is an indicator function, msg(t) is the content of the message at time t.

Furthermore, let the encrypted user traffic be $f^*(t)$ between the server and the client. The client can initiate multiple sessions, it starts with a handshake, therefore, in our model $h(t)$ is the aggregate handshake traffic between the server and the client. The encrypted user traffic is equal to the user traffic fragmented, encapsulated and encrypted to SSL/TLS record layer:

$$f^*(t) = \sum_{i=1}^{N} \left(\text{RHeader} + \text{Enc}\left\{\text{Compr}\left\{\text{msg}_i(t)\right\} + MAC\left\{\text{Compr}\left\{\text{msg}_i(t)\right\}\right\}\right\}\right),$$

where + is concatenation, Enc{.} means symmetric encryption, and $\text{msg}_i(t)$ is the i-th fragment of the message $\text{msg}(t) = \sum_{i=1}^{N}(\text{msg}_i(t))$ such as $N = \left\lceil \frac{|\text{msg}(t)|}{\varphi} \right\rceil$ and the φ is the size of SSL/TLS fragment.

As SSL/TLS can handle both stream ciphers $S\{.\}$ and block ciphers $B\{.\}$:

$$\text{Enc}\{.\} = B\{.\}I_B + S\{.\}I_S,$$

where I_x indicator function ($I_B I_S = 0$), its value is 1 if the specific cipher is used (otherwise zero). A short definition of block ciphers (let the $\vdash$ symbol be deduction, $|x|$ the length of message x in bytes and plaintext is $\alpha(t)$):

$$\begin{aligned}
\beta_B(t) &= B\{\alpha(t), \text{key}\} \\
&= B\{\text{key}, \alpha(t) \mapsto \beta_B(t) \mid \Pr(\beta_B(t) \vdash \alpha(t)) \\
&= 0 \wedge \Pr(\beta_B(t) \vdash \text{key}) \\
&= 0 \wedge |\beta_B(t)| \geq |\alpha(t)|\}
\end{aligned}$$

and Stream ciphers:

$$\begin{aligned}
\beta_S(t) &= S\{\alpha(t), \text{key}\} \\
&= B\{key, \alpha(t) \mapsto \beta_S(t) \mid |\alpha(t)| \\
&= |\beta_S(t)| \wedge \Pr(\beta_S(t) \vdash \alpha(t)) \\
&= 0 \wedge \Pr(\beta_S(t) \vdash \text{key}) = 0\}
\end{aligned}$$

Compr{.} means compression: $\beta_C = \text{Compr}\{\alpha(t)\} = \{\alpha(t) \mapsto \beta_C(t) \mid |\beta_C(t)| \leq |\alpha(t)|\}$ The RHeader is the general SSL/TLS record layer header, it consists of the type, version and length fields. MAC{.} is the message authentication code calculation function, this is different in SSL and TLS, but the following definition

can be applied to both protocols ($\gamma(t)$ is an arbitrary bit string):

$$\begin{aligned}\beta_{\mathrm{MAC}} &= \mathrm{MAC}\{\alpha(t), key\} \\ &= \{key, \alpha(t) \mapsto \beta_{\mathrm{MAC}}(t) \mid \Pr(\beta_{\mathrm{MAC}}(t) \vdash \alpha(t)) \\ &= 0 \wedge \Pr(\beta_{\mathrm{MAC}}(t) \vdash \mathrm{key}) \\ &= 0 \wedge (\Pr(\alpha(t), \mathrm{key} \vdash \gamma(t)) = 0 \mid \beta_{\mathrm{MAC}}(t) \neq \gamma(t))\}\end{aligned}$$

The generalised https model consists of the encrypted user application data and handshake traffic:

$$\mathrm{HTTPS}(t) = h(t) + f^*(t).$$

Consider $\underline{\xi}^{(i)} = (\xi_{\lambda_i})$, a vector of probability variable $\underline{\xi}^{(i)} : \Omega \mapsto \Re^i$. The first element refers to the delay of λ_1 message arrived relative to a reference time. $\xi_{\lambda_i}, i \neq 1$ are the probability variables of the interarrival times of message λ_i and λ_{i+1}.

If no session caching is used, then $i \in [1..14]$ and

$$^{(\mathrm{noSKC})}\underline{\lambda}^T = (\mathrm{HelloRequest}, \mathrm{ClientHello}, \mathrm{ServerHello}, \mathrm{ServerCert}, \mathrm{ServerKeyExchange}, \mathrm{CertRequest}, \mathrm{ServerHelloDone}, \mathrm{ClientCert}, \mathrm{ClientKeyExchange}, \mathrm{ClientCertificateVerify}, \mathrm{ClientChangeCipherSpec}, \mathrm{ClientFinished}, \mathrm{ServerChangeCipherSpec}, \mathrm{ServerFinished}).$$

If Session caching is set $i \in [1..6]$ and

$$^{(\mathrm{SKC})}\underline{\lambda}^T = (\mathrm{clientHello}, \mathrm{ServerHello}, \mathrm{ServerChangeCipherSpec}, \mathrm{ServerFinished}, \mathrm{ClientChangeCipherSpec}, \mathrm{ClientFinished}).$$

The handshake traffic can be divided into client-to-server $h_c(t)$ and server-to-client $h_s(t)$ directions. (*Fig. 1*):

$$h_c(t) = \sum_{i \in K} \left(h_{\lambda_i} I(\xi_{\lambda_i} < t)\right); \qquad h_s(t) = \sum_{i \in L} \left(h_{\lambda_i} I(\xi_{\lambda_i} < t)\right);$$

where $K_{\mathrm{SKC}} = \{1, 5, 6\}$ and $L_{\mathrm{SKC}} = \{2, 3, 4\}$ at session reuse, and $K_{\mathrm{noSKC}} = \{2, 8, 9, 10, 11, 12\}$ and $L_{\mathrm{noSKC}} = \{1, 3, 4, 5, 6, 7, 13, 14\}$ when https not uses previous session information. h_{λ_i} is the handshake message λ_i, $I(.)$ is an indicator function, its value is set when the corresponding message is sent.

4.1. Format of Handshake Messages

In this Section the formal description of the handshake messages will be introduced. The RHeader is the general SSL/TLS record layer header, it consists of the type, version and length fields, while the HHeader is the general SSL/TLS handshake header, it contains the handshake type, and length fields. Ver is the version attribute

of the sender. I_x is a generic indicator function, its value is one if and only if message x exists, otherwise zero.

The Record layer adds the record layer header to all upper layer protocol (handshake, application data) messages. If a message is supposed to be encrypted, then after fragmentation the $\Psi(.) = \text{RHeader} + \text{Enc}\{\text{Compr}\{.\} + \text{MAC}\{\text{Compr}\{.\}\}\}$ transformation is applied to it. HelloRequests consist only of the handshake layer header $h_{\text{helloReq}} = \text{HHeader}$. ClientHello contains HHeader, Ver, and a Rand random data.

$$h_{\text{clienthello}}(t) = \text{HHeader} + \text{Ver} + \text{Rand} + \text{CSessionID} + \text{CCipherSuites} + \text{Compression},$$

Where CSessionID, CCipherSuites, Compression attributes are the session ID of the client (variable length random ID), list of ciphersuites and list of compression methods offered by the client. In every implementation those fields additionally contain length fields even if this is not specified in the SSL/TLS standards. The server answers to the ClientHello message, choosing ciphersuite and compression method:

$$h_{\text{serverhello}}(t) = \text{HHeader} + \text{Ver} + \text{Rand} + \text{SSessionID} + \text{SCipherSuite} + \text{Compression}$$

The server certificate message contains a server specific certificate or certificate list. CertificatesLength indicates the length of the certificate, this variable is not in the SSL/TLS specification.

$$h_{\text{SCert}}(t) = \text{HHeader} + \text{CertificatesLength} + \text{CertList(server)}$$

In the Server Key Exchange message, if RSA key transport is chosen in the ciphersuite, RSA parameters, if Diffie–Hellman key exchange is selected, then DH parameters are sent. The message should also contain a signature to authenticate the traffic:

$$h_{\text{SKeyXch}}(t) = \text{HHeader} + \text{Ver} + I_{\text{RSA}}\text{RSA} + I_{\text{DHE}}\text{DH} + I_{\text{Sig}}\text{Sig},$$

where

$$I_{\text{RSA}} I_{\text{DHE}} = 0.$$

DH contains the Diffie-Hellmann public variables, g, n and Y_s, RSA [7] contains the modulus and the exponent. I_{Sig} indicates whether a signature exists, Sig is a signature algorithm, specified in the SCipherSuite. If DSS [8] authentication exists, an SHA-1 [9] hash, if the signature is based on RSA, then an MD5+SHA1 hash is applied:

$$\text{Sig} = I_{\text{RSA}}(\text{MD5} + \text{SHA}) + I_{\text{DSS}}(\text{SHA})$$

The server can optionally ask for client certificate for mutual authentication. In this case, the known certificate types (ClientCertTypes) and the known certificate

authority names (CAname) are specified. In addition, implementations also contain one byte long attribute for indicating the number of known certificate types (CertTypesCount).

$$h_{\text{CertReq}}(t) = \text{HHeader} + \text{CertTypesCount} + \text{ClientCertTypes} + \text{CAname}.$$

The ServerHelloDone consists of:

$$h_{\text{SHelloDone}}(t) = \text{HHeader}.$$

Client certificate is similar to the server certificate message, however, it contains the Client certificate or certificate list:

$$h_{\text{CCert}}(t) = \text{HHeader} + \text{CertificatesLength} + \text{CertList(client)}.$$

Client Key exchange message: if RSA key transport is selected, it contains Version and random number encrypted by the public RSA key found in the certificate. If DH method is selected, the public DH value is attached:

$$h_{\text{CKeyXch}}(t) = \text{HHeader} + I_{\text{RSA}}\{\text{Ver} + \text{RND}\}\text{RSA}_{\text{Pub}} + I_{\text{DHE}}Y_{\text{sDH}}.$$

The optional Client Certificate Verify message includes either DSS or RSA signature:

$$h_{\text{ClCertVerify}}(t) = \text{HHeader} + I_{\text{Sig}}\text{Sig},$$

The handshake phase is closed by the Finished message, generally this message is encrypted by the negotiated algorithms. This message contains verification data, which consists of 36 bytes in the case of SSL and 12 bytes in TLS.

$$h_{\text{Finished}}(t) = \text{HHeader} + \text{VerifyData}.$$

In SSL/TLS there are two additional non-handshake messages, the Alert and Change CipherSpec message. The former indicates warning or fatal error, while the latter indicates when the negotiated algorithms should be used. (Both can be in encrypted form):

$$h_{\text{Alert}}(t) = \text{AlertLevel} + \text{AlertDescription}$$

and

$$h_{\text{ChCipherSpec}}(t) = \text{CipherSpecType}.$$

5. Modelled Traffic Characteristics of https

In this chapter Steps 2,3,4,5 and 8 of the methodology for modelling traffic characteristic of security protocols are shown.

5.1. Cryptographic Algorithms

In the following, *ciphers used in SSL/TLS implementations* are shown: RSA is used for Authentication and Key transport, but also DH or DHE is used for key generation and DSS for authentication. For symmetric encryption 3DES, DES, RC4 and IDEA are used, and MD5 and SHA-1 for message integrity.

Block size or output length of the used cryptographic algorithms are the following: MD5 uses 16, SHA-1 uses 20 bytes. Block ciphering algorithms like DES, 3DES, IDEA have 8 bytes block length, RSA generates as long ciphertext as the size of modulus [10] so 64, 128, 256 bytes (if the key length is 512, 1024, 2048 bits). Output length of RC2, RC4 algorithms is the same as the length of the input plaintext.

Size of the cryptographic parameters that can be sent between the client and the server are various. The modulus of RSA can be around 64, 128, 256 bytes (512, 1024, 2048 bits), however, the RSA exponent is usually 3 bytes. The length of Diffie-Hellmann generator is usually 1 byte, while the modulus and Y_s typically have 64, 96, 128 bytes (512, 768, 1024 bits) length. A DSA signature (containing the r and the s) requires 40 bytes, while RSA signature requires at least 64 or 128 bytes (if key length is 512, 1024 bits, respectively).

5.2. Length of Message Variables

Constant length variables of the SSL/TLS messages and their sizes are the following: RHeader 5, HHeader 4, Ver 2, Rand 32, CertificatesLength 3, CertTypeCount, AlertLevel, AlertDesc, ChCipherSpecType and ComprMethod 1 byte.

Variable length message attribute is only the client SessionID data, whose length can be between 0 and 32 bytes, but in our measurement it is either 0 or 32.

Variable length message attributes which depend on some security properties are the following. The CipherSuiteList variable of ClientHello contains between 5 and 24 ciphersuite, however, in most cases 8 and 11 ciphersuites are sent to the server (one half of the samples contained 11 and one quarter of the samples had 8 ciphersuites). Since one ciphersuite consumes 2 bytes, the length of Ciphersuite variable can be modelled as: $\Pr(x < 0.5)22 + \Pr(0.5 < x < 0.75)16 + \Pr(x > 0.75)\ \mathrm{Unif}(10, 48)$.

The length of the server and the client certificate depends on several parameters, for example the number of X.509 extensions and signature algorithm. In addition, the length of certificate depends on the length of the bounded public key and signature. Public keys used on the Internet have 512, 1024 or 2048 bits length, however, the latter is not typical yet. Therefore the length of the Certificate payload is modelled as a uniform distribution between 600 and 1000 bytes.

The RSA public modulus has the same length as the key size, so it can be 64, 128 bytes. The RSA public exponent usually has a 3-bytes-length. Diffie-Hellman generator is usually 1 byte, while the modulus and Y_s can be 64, 96, 128 bytes.

The length of Signature data (Sig) can be 0, (in the case of no signature) 40, (if the signature algorithm is DSS) or 64, 128 (in the case of 512 or 1024 bits length RSA) bytes.

The Client CertTypesCount in the Certificate Request can be between 1 and 255 bytes, but in our measurement this is between 30 and 200. Therefore let the length of ClientCertTypes be modelled as uniform distribution between 30 and 200 bytes.

The RSA encrypted message attribute in the Client Key Exchange message ($\{\text{Ver} + \text{RND}\}\text{RSA}_{\text{Pub}}$) has the same length as the RSA modulus size so 64 or 128 bytes. In the Finished message, the Verification data is either 36 bytes, if SSL is used, or 12 bytes in the case of TLS.

5.3. *Length of Messages*

The traffic characteristic of https can be defined using the generalised https model:

$$|\text{HTTPS}(t)| = |h(t)| + |f^*(t)|,$$

Handshake messages can be divided into client and server communication,

$$|h(t)| = |h_c(t)| + |h_s(t)|,$$

where

$$|h_c(t)| = \sum_{i \in K} \left(|h_{\lambda_i}| I(\xi_{\lambda_i} < t) \right);$$

$$|h_s(t)| = \sum_{i \in L} \left(|h_{\lambda_i}| I(\xi_{\lambda_i} < t) \right);$$

Let $\underline{\phi}^{(i)} = (\phi_{\lambda_i})$, $\underline{\phi}^{(i)} : \Omega \mapsto \Re^i$ probability variable for the length of the λ_i message. In this case the handshake traffic length:

$$|h_c(t)| = \sum_{i \in K} \left(\phi_{\lambda_i} I(\xi_{\lambda_i} < t) \right);$$

$$|h_s(t)| = \sum_{i \in L} \left(\phi_{\lambda_i} I(\xi_{\lambda_i} < t) \right);$$

Furthermore, let the probability density function of ξ_{λ_i}: $f_{\lambda_i}(x)$. Suppose that $f_{\lambda_i}(x)$ is time invariant. Note that if $f_{\lambda_i}(x)$ and the distribution of ϕ_{λ_i} is known, then the handshake traffic can be modelled. In the following, several models for $\phi_{\lambda_i}(x)$ and for its distribution are presented, while models for $f_{\lambda_i}(x)$ are discussed in chapter 7.2.

The length of Record and Handshake layer header is fixed, 5 and 4 bytes, respectively ($|\text{RHeader}| = 5$, $|\text{HHeader}| = 4$). Let notation ($\{a, b, c\}$) be a specific

distribution between the set of a, b and c. The general model of Table 1 shows the length of handshake messages using parameter estimation from Section 5.2.

Record layer encryption is used to protect user data, finish and alert messages.

$$|f^*(t)| = \sum_{i=1}^{N} (|\text{RHeader}| + |B\{\text{payload}\}|I_B + |S\{\text{payload}\}|I_S),$$

where the payload before encryption:

$$\text{payload} = \text{Compr}\{\text{msg}_i(t)\} + \text{MAC}\{\text{Compr}\{\text{msg}_i(t)\}\}.$$

The MAC algorithm is calculated as the outmost function is a hash. Therefore the length of MAC depends on the hash algorithms: $|\text{MAC}\{.\}| = I_{\text{MAC}}(16I_{\text{MD5}} + 20I_{\text{SHA}})$ The block ciphered data length:

$$|B\{\text{payload}\}| = |\text{Compr}\{\text{msg}_i(t)\}| + |\text{MAC}\{\text{Compr}\{\text{msg}_i(t)\}\}| + |\text{Padding}|.$$

Let the block length be k bit. If the communication uses no traffic flow confidentiality (so the padding fills out only the last block), then the padding length can be modelled as a uniform distribution: $|\text{Padding}| = \text{Unif}(0, k/8)$. Otherwise, if traffic flow confidentiality is set, then the padding can fill at least the last data block, but its length can reach 255 bytes:

$i \cdot \text{blocksize} + \text{Unif}(0, k/8) \leq 255$, where i is a probability variable, such $i \in \aleph$ and its minimal value is $\text{Univ}(0, k/8)$ and maximal value is 255. The distribution of this probability variable can be modelled with uniform distribution $i = \text{Unif}(\text{Unif}(0, k/8), 255)$.

For example, block ciphers with block size of 64 bits and with no traffic flow confidentality, $|\text{Padding}| = \text{Unif}(0, 8)$. This is the common case in our measurements.

The stream ciphered data length:

$$|B\{\text{payload}\}| = |\text{Compr}\{\text{msg}_i(t)\}| + |\text{MAC}\{\text{Compr}\{\text{msg}_i(t)\}\}|.$$

5.4. *Simplifications in the Model*

In the *First refinement model* we made several improvements to adjust our model to the common properties of the implementations. Therefore the size of the Client Hello and ServerHello message is increased by four and one, respectively. The implementations use extra length fields in those messages. The Certificate of the client and the server is modelled by a uniform distribution between 600 and 1000 bytes, because the length of all the certificates was between those values.

The *Second refinement model* contains simplifications of various algorithms. In all measurements no compression is used by implementations, so

$$|\text{Compr}\{f(t)\}| = |f(t)|.$$

In our measurements RSA_RC4_128_MD5 was the most frequently used ciphersuite (89.5%), in most implementations it is included, and it has the highest preference. Occurrence of RSA_RC4_56_MD5 is 1.7%, RSA_RC4_128_SHA is 0.35%, RSA_RC4_56_SHA is 8.2%. The remaining block ciphers, (like DHE_RSA_3 DES_SHA) are used only in 0.25% of the samples. Therefore we can conclude that in our measurements the stream ciphers have dominance (99.75%), therefore let us simplify our model as $I_B = 0$, $I_S = 1$.

In most cases (99.8%) RSA is used, so let us simplify our model further, as $I_{\text{RSA}} = 1$, $I_{\text{DSS}} = 0$, $I_{\text{DH}} = 0$, $I_{\text{DHE}} = 0$. From the RSA ciphersuites 1.8% was RSA_EXPORT (with 512 bits), 8.2% was RSA_EXPORT_1024 (with 1024 bits) and 90% of the ciphersuites was RSA with no limitation. However, unlimited RSA ciphers used only 1024 bits, therefore the 1024-bit-long RSA had the dominance (98.2%). SHA algorithm is used in 8.8% of the https connection, while MD5 is commonly used 91.2%. All communication used integrity check, so $I_{\text{MAC}} = 1$.

SSL is used in 52% of the https traffic (TLS is used in 48% of communication). Therefore we simplify that the length of finished message is (with 50% probability) either 45 or 21 bytes. In ServerKeyExchange and in ClientVerify messages SHA signature is used.

The SessionID in the ClientHello is set when an SSL/TLS session has been set up previously, and no new SSL/TLS connection is forced. Therefore the usage of SessionID is set the case of session caching.

The modified length of handshake messages can be found in *Table 1*.

Note that the calculation of the record layer length is simplified ($N = \left\lceil \frac{|\text{msg}(t)|}{\varphi} \right\rceil$):

$$|f^*(t)| = \sum_{i=1}^{N} \left(|\text{RHeader}| + |\text{msg}_i(t)| + I_{\text{MD5}} |\text{MAC}\{\text{msg}_i(t)|\} \right)$$

$$= \sum_{i=1}^{N} \left(|\text{RHeader}| + I_{\text{MD5}} |\text{MAC}\{\text{msg}_i(t)|\} \right) + |f(t)| = 21N + |f(t)|.$$

In the *Third refinement model* unfrequent messages (with an occurrence under 0.5%) are removed (*Table 1*). Since all optional handshake messages are removed, only the ClientHello, ServerHello, SCert, SHelloDone, ClKeyExchange, ChCipherSpec, Finished messages remain.

The *Session Key Caching* technique requires less messages, the length of those messages can be found in *Table 1*.

6. Behaviour of User and Network

The nature of the https traffic highly depends on user behaviour. However, https services, like Internet banking, e-commerce, secure mailing, etc. delimit the behaviour of the user. Therefore three different types of secure http service can be

easily separated. In the first type the user interactively uses the service, reads and fills https forms and sends them back (e.g. a questionnaire). In the second type the user primarily reads information, and seldom sends data (e.g. querying an on-line bank). The third type of secure web service is a secure mailer, where the user reads and sends mails. In this case the user can attach a large file for uploading, or the user may also download large files.

The user-generated traffic is correlated to the handshake traffic. Every new standalone HTTP object download precedes a new handshake message flow.

The secure layer opens a new secure connection for each new object. Therefore if the session caching is not set, then the https traffic:

$$\mathrm{HTTPS}(t) = \sum_{j=1}^{\#\mathrm{objects}} \sum_{i=1}^{N} \big(h_{\mathrm{noSKC},i,j}(t) + \mathrm{RHeader}_{i,j} + \mathrm{msg}_{i,j}(t) + \mathrm{padding}_{i,j} + \mathrm{MAC}_{i,j}\big).$$

In session key caching is used, the download of the first object uses full handshake (because there is no preceding shared knowledge), however further objects use session caching mechanisms:

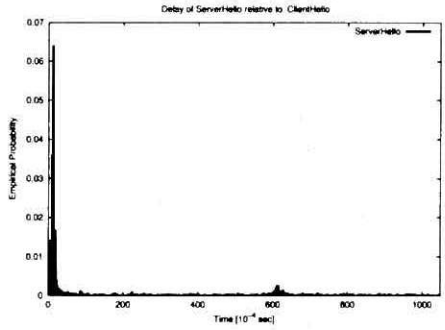
$$\mathrm{HTTPS}(t) = h_{\mathrm{noSKC},1}(t) + \sum_{i=1}^{N} (\mathrm{RHeader}_i + \mathrm{msg}_{i,1}(t) + \mathrm{padding}_{i,1} + \mathrm{MAC}_{i,1} + \sum_{j=2}^{\#\mathrm{objects}} \sum_{i=1}^{N} \big(h_{\mathrm{SKC},i,j}(t) + \mathrm{RHeader}_{i,j} + \mathrm{msg}_{i,j}(t) + \mathrm{padding}_{i,j} + \mathrm{MAC}_{i,j}\big).$$

The network also can distort the traffic shape of https; buffering delays, network congestion, the state of TCP (e.g. in slow start), propagation delays may cause significant deviations.

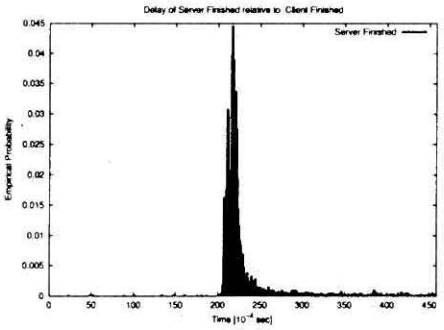
However, in our study the delay is corrected by the actual RTT. Therefore effects from the network layer can be ignored.

7. Results

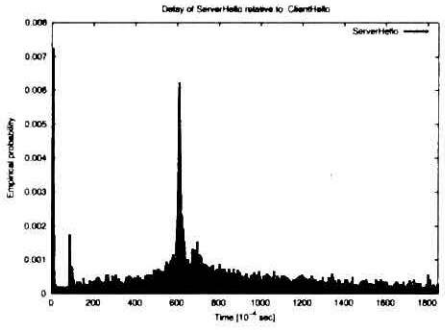
Our measurements have been performed in two scenarios. In the first configuration we monitored a 100 Mbps backbone line. The second measurement was collected on the 10 Mbps network segments where https server was located. In both scenarios we monitored real https traffic, which sent on TCP port 443. In the first configuration more than 120000, while in the second configuration more than 20000 https communications (containing handshake and user data traffic) were collected.



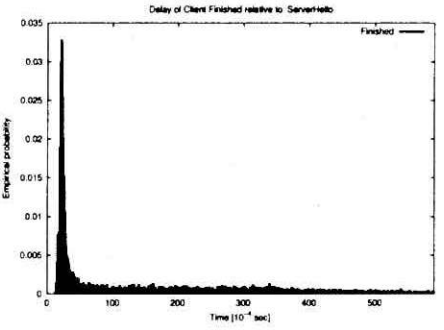
(a) Server Hello messages, No SKC



(b) Server Finished messages, No SKC



(c) Server Hello messages, SKC



(d) Finished messages, SKC

Fig. 2. Empirical distributions of interarrival times

7.1. Length of HS Messages

We have four methods to estimate the length of handshake messages. The first one is originated from the general traffic characteristics in Section 5.3. In this model there are several unresolved variables that the first refinement model resolves. This might give a rough estimation, but if it is not sufficient enough, then the second refinement model can be used, where the unfrequent variables are removed. In the third refinement model unfrequent messages are also neglected. The SKC model can only be used for connections using session key caching mechanism (*Table 1*).

Table 1. Length of handshake messages. * length includes the Record layer encapsulation

Handshake messages (λ_i)	Empirical	general model ϕ_{λ_i}	refinement model1	refinement model2	refinement model3	SKC
Hello Req*	9	9	–	9	–	–
ClientHello*	$p = 0.5 : 102$ $p = 0.25 : 96$ $p = 0.25 :$ Unif(45, 110)	$44 + 32I_{\text{sessionID}}$ +2(#Ciphersuite)	$48 + 32I_{\text{SKC}}$ +2({8, 11})	({96, 102})	({96, 102})	({96, 102})
Server Hello	74	73	74	74	74	74
Server Cert	Unif(600,1000)	$\phi_{\text{cert}} = \phi_{\text{cert}}(\text{server})$	12+ Unif(600, 1000)	806	806	–
ServerKeyExchange	210	$11 + I_{RSA}(3$ +({64, 128}))+ $I_{\text{DHE}}(1$ +2({64, 96, 128})) $+I_{\text{Sig}}(\{40,$ 64, 128}))	–	180	–	–
S Hello Done	4	4	–	4	4	–
Client Cert	Unif(600,1000)	ϕ_{cert} $= \phi_{\text{cert}}(client)$	12+ Unif(600, 1000)	806	806	–
ClientKeyExchange	127	$9 + I_{RSA}(3$ +({64, 128})) $+I_{DHE}(1$ +({64, 96, 128}))	–	140	140	–
Cert Verify	37	$9 + I_{\text{Sig}}(\{40$, 64, 128})		49	–	–
ChCipherS*	6	6	–	6	6	6
Finished*	({45, 21})	9 + ({36, 12})	–	({45, 21})	–	–
Alert*	7	7	–	7	7	7
Encrypted Finished*	$p = 0.067$: 37 $p = 0.015$: 41 $p = 0.869$: 61 $p = 0.047$: 65	{21, 45} +Unif(0, 8) +|MAC|	–	({37, 41, 61, 65})	({37, 41, 61, 65})	61
Encrypted Alert*	$p = 0.802$: 23 $p = 0.191$; 27	7 + Unif(0, 8) +|MAC|	–	23	23	23
Record Layer	–	|Compr{$\text{msg}_i(t)$}| +Unif(Unif(0, 8), 255) +|MAC{Compr{$\text{msg}_i(t)$}}|		$21 + \|f(t)\|$	$21 + \|f(t)\|$	NA

7.2. *Time Dependence*

The lower layer of the SSL/TLS protocols optimises the handshake performance by grouping successive handshake messages. The empirical density of the interarrival times of the grouped messages are shown in *Fig. 2*. The distortion caused by network is corrected by the actual RTT, furthermore the data was collected in the second scenario, in the same LAN where the server resides. *Fig. 2(a)* demonstrates that the server immediately (2–3 ms) responds to ClientHello. In some cases, however,

the server looks up its database for the client. This causes an extra delay (60 ms) which is significant in the case of session caching *Fig. 2(c)*.

The delay between the two finished messages (20 ms) in non-session key caching case (*Fig. 2(b)*) is significant compared to the session caching case. The cause of this difference can be found in the key agreement mechanisms, since the server processes the CPU intensive ClientKeyExchange message, while in the session key caching case this step is replaced by a faster one.

Note that the empirical probability density functions can be used for estimating the density function of ξ_{λ_i}.

7.3. *Overhead of https*

Handshake overhead

The handshake overhead is defined as the ratio of the handshake and the total traffic of a connection.

$$\begin{aligned} R_{\text{handshake}} &= \frac{|h(t)|}{|h(t)| + |f^*(t)|} \\ &= \frac{|h(t)|}{|h(t)| + N(|\text{RHead}| + |\text{padding}| + |\text{MAC}(f(t))|) + |f(t)|} \\ &= \frac{|h(t)|}{|h(t)| + N(|\text{RHead}| + |\text{padding}| + |\text{MAC}(f(t))|) + N\varphi - \nu}, \end{aligned}$$

where $\nu = 0..(\varphi - 1)$.

The maximal value of handshake overhead (when no padding, MAC):

$$R_{\text{handsake}_{\text{MAX}}} = \frac{|h(t)|}{|h(t)| + 5N + |f(t)|} = \frac{|h(t)|}{|h(t)| + 6}$$

when $N = 1$, $\nu = \varphi - 1$. The minimal value of the handshake overhead (block cipher with 64 blocklength, SHA):

$$R_{\text{handsake}_{\text{MIN}}} = \frac{|h(t)|}{|h(t)| + |f(t)| + 33N} = \frac{|h(t)|}{|h(t)| + N(33 + \varphi)}.$$

The most possible overhead value (the ciphersuite TLS_RSA_RC4_MD5):

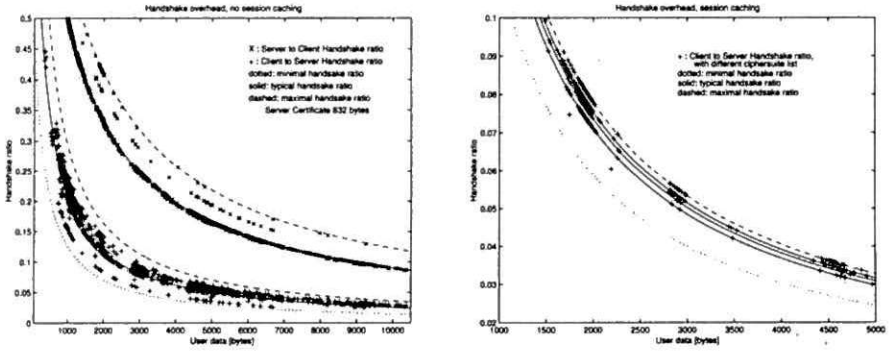
$$R_{\text{handsake}} = \frac{|h(t)|}{|h(t)| + |f(t)| + 21N} = \frac{|h(t)|}{|h(t)| + N(21 + \varphi) - \nu}.$$

Table 2 shows the maximal and minimal values of handshake communication using the models in *Table 1*.

Fig. 3 shows how the handshake ratio depends on the user traffic. The dotted and dashed lines are the minimal and maximal borderlines of the handshake ratio.

Table 2. Length of handshake messages and overhead. * for 1kbyte user data

	Session Caching			Non Session Caching		
Handshake traffic	Min	Max	Typical	Min	Max	Typical
Client to Server	125	261	169	159	369	264
Server to Client	122	150	146	131 + cert	369 + cert	155 + cert
Overhead						
Client to Server	$\frac{125}{125+N(33+\varphi)}$	0.9775	0.1392*	$\frac{159}{159+N(33+\varphi)}$	0.984	0.201*
Server to Client	$\frac{122}{122+N(33+\varphi)}$	0.9615	0.1225*	$\frac{131+\text{cert}}{131+\text{cert}+N(33+\varphi)}$	$\frac{1-6}{375+\text{cert}}$	$\frac{1-1045}{1200+\text{cert}}$ *



(a) No session caching (b) Session Caching

Fig. 3. Handshake ratio (empirical)

In *Fig. 3(a)* the server to client and the opposite direction of handshake ratio are represented. *Fig. 3(b)* proves that even the ciphersuite list affects the handshake ratio, in the figure the ClientHello contained 3,5,6,8,11 ciphersuites, thus the handshake ratio increases.

Cost for security

The traffic cost for securing http is defined as the ratio of the sum of the

SSL/TLS control messages, record layer overhead and the user traffic.

$$\eta = \frac{|h(t)| + N(|\text{RHead}| + |\text{MAC}(f(t))| + |\text{padding}|)}{|f(t)|}$$
$$= \frac{|h(t)| + N(|\text{RHead}| + |\text{MAC}(f(t))| + |\text{padding}|)}{N\varphi - \nu},$$

The minimal cost line (when there is no padding and no MAC).

$$\eta_{\min} = \frac{|h_{\min}(t)|}{N\varphi} + \frac{5}{\varphi},$$

The maximal cost line (blockcipher, SHA MAC, etc.)

$$\eta_{\max} = \frac{|h(t)_{\max}| + 33N}{|f(t)|} = |h(t)_{\max}| + 33.$$

It is obvious that the cost function depends on the user traffic. If the user can estimate the amount of traffic he sends , the additional network traffic can be determined. Therefore this information can be the basis of the network dimensioning, or bandwidth estimation.

8. Conclusion

Https model can be used not only for describing secure web traffic, but it can be used for characterisation of secure IMAP, and secure POP3 [12] also. User traffic and network congestion can distort the shape of https, however, the user behaviour is determined by the secure service. In this paper a general https model, and an exact traffic model together with simplified traffic models are introduced.

References

[1] FRIER – KARLTON – KOCHER, *The SSL 3.0 Protocol*, Netscape Corp., 1996. Internet Draft, Work in Progress.
[2] KIPP, E. – HICKMAN, B., *The SSL Protocol*, Netscape Corp., 1995. Internet Draft, Work in Progress.
[3] DIERKS, T. – ALLEN, C., *The TLS Protocol*, RFC 2246, 1999. Proposed Standard.
[4] BERNERS-LEE, T. – FIELDING, R. – FRYSTYK, H., *Hypertext Transfer Protocol – HTTP/1.0*, RFC 1945, May 1996.
[5] FIELDING, et al., *Hypertext Transfer Protocol – HTTP/1.1*, RFC 2616, 1999.
[6] RIVEST, R., *The MD5 Message-Digest Algorithm*, RFC1321, 1992.
[7] SCHNEIER, B., *Applied Cryptography*, 2nd Edition, John Wiley, 1996.
[8] FIPS PUB 186-2, National Institute of Standards and Technology, 2000.
[9] FIPS PUB 186-1, National Institute of Standards and Technology, 1995.
[10] Public Key Cryptography Standards PKCS#1, RSA laboratories, 1998.
[11] RESCORLA, E., *HTTP Over TLS*, RFC2818, 2000.
[12] NEWMAN, C., Using TLS with IMAP, POP3 and ACAP, RFC2595, 1999.