



UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE MATEMÁTICA
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO



Eliézio Batista de Oliveira

Implementação das Extensões do Protocolo TLS no OpenSSL

Rio de Janeiro – RJ

Março / 2006

Eliézio Batista de Oliveira

`eliezio@computer.org`

Implementação das Extensões do Protocolo TLS no OpenSSL

Projeto Final de Curso submetido ao Departamento de Ciência da Computação do Instituto de Matemática da Universidade Federal do Rio de Janeiro como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Orientador:

Prof. Paulo Henrique de Aguiar Rodrigues, Ph.D.

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
INSTITUTO DE MATEMÁTICA
UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

Rio de Janeiro – RJ

Monografia do Projeto Final de Curso sob o título *Implementação das Extensões do Protocolo TLS no OpenSSL* , defendida por Eliézio Batista de Oliveira e aprovada em 31 de Março de 2006 pela banca examinadora constituída pelos professores:

Prof. Paulo Henrique de Aguiar Rodrigues, Ph.D.
Orientador

João Carlos Peixoto de Almeida da Costa, M.Sc.
NCE - UFRJ

Fabio David, M.Sc.
NCE - UFRJ

Ao meu pai (*in memoriam*)

Agradecimentos

Esse trabalho, fruto de uma gestação longa e intermitente, não seria possível sem o incentivo e o apoio decisivo de muitos. Meus especiais agradecimentos:

- ✧ Ao Deus Eterno, em quem descobri um pai amoroso.
- ✧ A minha esposa Genáina e meus filhos Felipe, Caio e Victor, meus incansáveis incentivadores, que aceitaram a minha ausência para realizarmos, juntos, esse sonho.
- ✧ Aos pastores Fernando Miranda, Luciano Vilaça e Moisés Fontoura, por me ajudarem a resgatar a minha humanidade e me mostrarem que ser homem é mover-se segundo o coração de Deus.
- ✧ A Alexandre Pi que com sua longanimidade ímpar me ensinou a buscar uma oportunidade em cada dificuldade, no trabalho e na vida.
- ✧ Ao meu orientador Paulo Aguiar pelo seu inestimável apoio, que muito me ajudou a não deixar de mirar o alvo desse projeto.
- ✧ A Dayse Lobo Cavalcanti e equipe pela disposição e paciência para resolver os inúmeros obstáculos nesse esforço final.

“If you think technology can solve your security problems,
then you don’t understand the problems
and you don’t understand the technology.”

Beyond Fear
BRUCE SCHNEIER

Resumo

Implementação das Extensões do Protocolo TLS no OpenSSL

O TLS, sucessor oficial do SSL, tem se consolidado como o protocolo de segurança mais utilizado na Internet, estando embutido em praticamente todos os navegadores e servidores HTTP.

A sua utilização em sistemas embutidos tem, entretanto, conflitado com os limitados recursos que estes equipamentos dispõem. A RFC 3546 apresenta algumas extensões ao TLS visando minimizar o ônus da sua adoção.

Este texto descreve a implementação destas extensões realizadas pelo autor no OpenSSL, um *toolkit* TLS/SSL de código aberto largamente utilizado em servidores HTTP.

Este texto apresenta também uma extensão adicional proposta e implementada pelo autor para reduzir ainda mais o volume de dados necessários para o estabelecimento de uma conexão TLS.

Abstract

Implementation of TLS Extensions in OpenSSL

TLS, the official successor of SSL, is becoming the mostly widely used security protocol on data communication networks, notably the Internet, virtually available on all web browsers and servers.

Its usage by embedded systems faces, however, some challenges due to the very restricted resources available to those systems. The IETF's RFC 3546 recommends some extensions especially targeted to minimize the protocol overhead.

This dissertation describes the implementation of these extensions made by this author by means of modifications on OpenSSL, an open-source TLS/SSL toolkit commonly employed in HTTP servers.

Furthermore, this text presents an additional extension advised and implemented by this author alongside the official extensions, aimed to reduce the volume of network traffic during the TLS connection establishment.

Sumário

Figuras	12
Tabelas	13
Listagens	14
Siglas	16
Introdução	18
Capítulo 1: Introdução à Criptografia	19
1.1 Funções de <i>Hash</i> (resumo criptográfico)	21
1.2 Algoritmos de Chave Secreta	21
1.2.1 <i>Message Authentication Code</i>	22
1.2.2 Cifradores Simétricos	22
1.3 Algoritmos de Chave Pública	23
1.3.1 Cifragem	24
1.3.2 Assinatura Digital	24
1.3.3 Estabelecimento de Chave	25
1.4 Certificados Digitais	25
Capítulo 2: Introdução ao protocolo TLS	27
2.1 O criptossistema TLS	27
2.2 Conexão e Sessão TLS	27
2.3 Arquitetura do protocolo TLS	28
2.3.1 <i>TLS Record Protocol</i>	29
2.3.2 <i>Alert Protocol</i>	30
2.3.3 <i>Handshake Protocol</i>	30

Capítulo 3: As extensões do protocolo TLS	35
3.1 <i>Server Name Indication</i> (SNI)	36
3.2 <i>Maximum Fragment Length</i> (MFL)	36
3.3 <i>Client Certificate URL</i> (CCU)	37
3.4 <i>Trusted CA Indication</i> (TCI)	37
3.5 <i>Truncated HMAC</i> (TMAC)	38
3.6 <i>Certificate Status Request</i> (CSR)	38
3.7 <i>Server Certificate Chain Indication</i> (SCCI)	38
Capítulo 4: Implementação das extensões	40
4.1 Diretrizes de desenvolvimento	40
4.2 Modificações Gerais	41
4.2.1 <i>Build</i>	41
4.2.2 Estruturas de Dados	41
4.2.3 Funções Nativas	42
4.2.4 Novas Funções	43
4.3 Testes de Conformidade e Interoperabilidade	43
4.3.1 Aplicações de teste	43
4.3.2 Ambiente de teste	45
4.3.3 Testes de Interoperabilidade	46
4.3.4 Apresentação dos resultados dos testes	46
4.4 <i>Server Name Indication</i>	48
4.4.1 Especificação	48
4.4.2 Divergências	48
4.4.3 API Estendida	48
4.4.4 Implementação	49
4.4.5 Testes de Conformidade	49
4.4.6 Testes de Interoperabilidade	52
4.5 <i>Maximum Fragment Length</i>	54

4.5.1	Especificação	54
4.5.2	Divergências	54
4.5.3	API Estendida	54
4.5.4	Implementação	54
4.5.5	Testes de Conformidade	55
4.6	<i>Client Certificate URL</i>	59
4.6.1	Especificação	59
4.6.2	Divergências	60
4.6.3	API Estendida	60
4.6.4	Implementação	61
4.6.5	Testes de Conformidade	62
4.7	<i>Truncated HMAC</i>	67
4.7.1	Especificação	67
4.7.2	API Estendida	67
4.7.3	Divergências	67
4.7.4	Implementação	67
4.7.5	Testes de Conformidade	68
4.8	<i>Trusted CA Indication</i>	71
4.8.1	Especificação	71
4.8.2	API Estendida	72
4.8.3	Divergências	72
4.8.4	Testes de Conformidade	72
4.9	<i>Server Certificate Chain Indication</i>	74
4.9.1	Especificação	74
4.9.2	API Estendida	74
4.9.3	Divergências	75
4.9.4	Implementação	75
4.9.5	Testes de Conformidade	75

Capítulo 5: Conclusão e Trabalhos Futuros	78
5.1 Conclusão	78
5.2 Propostas de Trabalhos Futuros	79
5.2.1 Tarefas Gerais	79
5.2.2 Extensão <i>Server Name Indication</i>	79
5.2.3 Extensão <i>Client Certificate URL</i>	79
5.2.4 Extensão <i>Certificate Status Request</i>	80
5.2.5 Extensão <i>Server Certificate Chain Indication</i>	80
Apêndice A: Header “tlsx.h”	81
Apêndice B: Programa “MakePkiPath.java”	84
Referências Bibliográficas	86

Figuras

1	Ferramentas criptográficas e suas inter-correlações	20
2	Geração e validação do MAC	22
3	Cifragem e decifragem usando SKA	23
4	Cifragem e decifragem com PKA	24
5	Assinatura Digital com PKA	24
6	TLS na arquitetura em camadas de protocolos	28
7	<i>TLS Record Protocol</i>	29
8	<i>TLS Record Header</i>	29
9	<i>Handshake</i> completo com autenticação do servidor	31
10	<i>Handshake</i> completo com autenticação mútua	31
11	<i>Handshake</i> abreviado	33
12	Fragmentação da mensagem <i>Certificate</i>	56

Tabelas

1	Simbologia usada nas definições dos algoritmos	20
2	<i>TLS Record Header</i>	30
3	<i>Handshake</i> completo com autenticação do servidor	32
4	<i>Handshake</i> completo com autenticação mútua	33
5	<i>Handshake</i> abreviado	34
6	Comandos especiais do aplicativo <code>s_client</code>	45
7	<i>Handshake</i> com registros limitados a 1024 bytes	57
8	Conexão TLS com registros limitados a 1024 bytes	58
9	Diálogo entre as aplicações <code>s_client</code> e <code>s_server</code> para testar a eficácia da extensão TMAC	68
10	Primeira conexão (nova): Registros com HMAC reduzido	69
11	Primeira conexão (restaurada): Registros com HMAC reduzido	70

Listagens

1	Script <code>s_server.sh</code>	46
2	Script <code>s_client.sh</code>	46
3	RFC 3546, trecho da seção 3.1	48
4	API para a extensão SNI	48
5	<i>Callback</i> para tratar extensão SNI	49
6	Chamada do script de teste para a extensão SNI	49
7	Primeira Conexão – Decodificação da mensagem <i>Client Hello</i>	50
8	Primeira Conexão – Saída da aplicação <code>s_server</code>	50
9	Segunda Conexão – Decodificação da mensagem <i>Client Hello</i>	51
10	Segunda Conexão – Saída da aplicação <code>s_server</code>	51
11	Conexão oriunda do Opera – Decodificação da mensagem <i>Client Hello</i>	52
12	Resposta ao <i>request</i> do Opera 9 – Decodificação da mensagem <i>Server Hello</i>	53
13	RFC 3546, trecho da seção 3.2	54
14	API para a extensão MFL	54
15	Chamada do <code>tethereal</code> para o detalhamento dos registros TLS	55
16	Chamada do <code>s_client.sh</code> para o teste da extensão MFL	55
17	Mensagem <i>Client Hello</i> com a extensão MFL habilitada	55
18	Definição do novo tipo de mensagem de <i>handshake Certificate URL</i>	59
19	RFC 3546, trecho da seção 3.3	59
20	API Estendida para a extensão CCU	60
21	Conversão de certificados para o formato DER	61
22	Chamada da <code>s_server.sh</code> para forçar a autenticação do cliente	62
23	Chamada da <code>s_client.sh</code> para teste da extensão CCU	62
24	Chamada do <code>tethereal</code> para capturar tráfegos TLS e HTTP	62
25	Mensagem <i>Certificate URL</i>	62

26	Solicitação HTTP enviada pela aplicação <code>s_server</code>	63
27	Início (cabeçalho) da resposta HTTP enviada pelo servidor para a aplicação <code>s_server</code>	63
28	Mensagem <i>Certificate URL</i>	64
29	Solicitação HTTP enviada pela aplicação <code>s_server</code>	65
30	Início (cabeçalho) da resposta HTTP enviada pelo servidor para a aplicação <code>s_server</code>	65
31	RFC 3546, trecho da seção 3.5	67
32	API Estendida para a extensão TMAC	67
33	Detalhe do <i>Client Hello</i> estendido	68
34	RFC 3546, trecho da seção 3.4	71
35	API Estendida para a extensão TCI	72
36	Cálculo do <i>hash</i> SHA-1 do certificado do CA	73
37	Mensagem <i>Client Hello</i> contendo a extensão TCI	73
38	Definição (não-oficial) da extensão SCCI	74
39	Especificação (não-oficial) da extensão SCCI	74
40	API Estendida para a extensão SCCI	74
41	Chamada da <code>s_client.sh</code> para o teste da extensão SCCI	76
42	Mensagem <i>Client Hello</i> com a extensão SCCI	76
43	Resposta do servidor com destaque para a mensagem <i>Certificate vazia</i> . .	76
44	<i>Header</i> <code>tlsex.h</code>	81
45	Programa <code>MakePkiPath.java</code>	84

Siglas

API	<i>Application Programming Interface</i>
ASN.1	<i>Abstract Syntax Notation One</i>
CA	<i>Certification Authority</i>
CCU	<i>Client Certificate URL (Extensão TLS)</i>
CSR	<i>Certificate Status Request (Extensão TLS)</i>
DCS	<i>Data Certification Server</i>
DER	<i>Distinguished Encoding Rules</i>
DH	<i>Diffie-Hellman</i>
DN	<i>Distinguished Name</i>
DNS	<i>Domain Name System</i>
DoS	<i>Denial of Service</i>
DSS	<i>Digital Signature Standard</i>
GPRS	<i>General Packet Radio Services</i>
HMAC	<i>Hash-based Message Authentication Code</i>
HTTP	<i>Hypertext Transfer Protocol</i>
HTTPS	<i>HTTP sobre TLS/SSL</i>
IANA	<i>Internet Assigned Numbers Authority</i>
IETF	<i>Internet Engineering Task Force</i>
IPsec	<i>IP Security</i>
ITU	<i>International Telecommunication Union</i>
ITU-T	<i>ITU Telecommunication Standardization Sector</i>
MAC	<i>Message Authentication Code</i>
MD5	<i>Message Digest 5</i>

MFL	<i>Maximum Fragment Length (Extensão TLS)</i>
MIC	<i>Message Integrity Code</i>
MIME	<i>Multipurpose Internet Mail Extensions</i>
OCSP	<i>Online Certificate Status Protocol</i>
OCSP-X	<i>Online Certificate Status Protocol Extensions</i>
PDA	<i>Personal Digital Assistant</i>
PKA	<i>Public Key Algorithm</i>
PEM	<i>Privacy-Enhanced Electronic Mail</i>
RSA	<i>Rivest, Shamir and Adleman's public-key cryptosystem</i>
SCCI	<i>Server Certificate Chain Indication (Extensão TLS)</i>
SCVP	<i>Simple Certificate Validation Protocol</i>
SHA-1	<i>Secure Hash Algorithm 1</i>
SKA	<i>Secret Key Algorithm</i>
SNI	<i>Server Name Indication (Extensão TLS)</i>
SSL	<i>Secure Socket Layer</i>
TCI	<i>Trusted CA Indication (Extensão TLS)</i>
TCP	<i>Transport Control Protocol</i>
TLS	<i>Transport Layer Security</i>
TMAC	<i>Truncated HMAC (Extensão TLS)</i>
URL	<i>Uniform Resource Locator</i>
VPN	<i>Virtual Private Network</i>

Introdução

O protocolo *Transport Layer Security* (TLS) [1], previamente conhecido como *Secure Socket Layer* (SSL), tem se tornado o mecanismo mais popular para o estabelecimento de comunicações seguras na Internet, sendo usado inclusive como alternativa ao IPsec na criação de *Virtual Private Networks* (VPNs).

Entretanto, a sua adoção em sistemas embutidos tais como telefones celulares e PDAs enfrenta alguns desafios:

- Relativa baixa capacidade de processamento desses dispositivos. Os diversos algoritmos criptográficos usados no TLS demandam, por natureza, uma alta utilização de CPU;
- Sua relativa baixa capacidade de armazenamento, que pode limitar a sua capacidade de autenticação;
- Usam em geral meios de comunicação com banda limitada, especialmente em redes compartilhadas onde o tráfego de dados não é prioritário em relação às demais mídias, como é típico na rede GPRS;
- A alta latência inerente às redes via satélite;
- Algumas redes *wireless* são comumente tarifadas por volume de dados.

Visando torná-lo mais leve para equipamentos sujeitos a essas restrições, o *IETF TLS Working Group* publicou em Junho de 2003 a RFC 3546 [2], intitulado “*Transport Layer Security (TLS) Extensions*”, que propõe diversas extensões ao TLS, preservando a retrocompatibilidade com as versões pré-existentes do protocolo.

As reduções promovidas por essas extensões oficiais, entretanto, não tratam da redução da mensagem que contém o certificado do servidor, responsável por aproximadamente 70% do *overhead* do TLS no volume de dados durante o estabelecimento de uma conexão.

Face a esta lacuna, uma extensão adicional foi então concebida por este autor para tratar especificamente desta notável redução, aplicável a uma grande parte das conexões TLS.

O objetivo desse projeto final de curso é implementar esta extensão adicional e a maioria das extensões oficiais da RFC 3546 na pilha TLS/SSL de código aberto OpenSSL.

1

Introdução à Criptografia

A segurança provida pelo TLS é obtida através do uso de diversas técnicas criptográficas que, em conjunto, provêem os seguintes serviços de segurança conforme ilustrado na Figura 1 na página seguinte, adaptada de [3]:

Confidencialidade – É a garantia de que o conteúdo legível dos dados só estará acessível para as entidades autorizadas;

Integridade – É a garantia de que os dados não foram modificados inadvertidamente;

Autenticidade – É a garantia sobre a identidade da origem da informação¹.

Apresentaremos a seguir uma conceituação básica de cada uma dessas ferramentas. Para uma introdução mais completa recomendamos o texto “*Handbook of Applied Cryptography*” [4].

Como ainda não há um consenso na tradução para português de alguns termos básicos em criptografia, arbitramos pela seguinte terminologia nesse documento:

Inglês	Português
encryption	cifragem
decryption	decifragem
cipher	cifrador
plaintext	mensagem ou texto legível
ciphertext	texto cifrado

Nas descrições dos algoritmos empregaremos a simbologia expressa na Tabela 1 na próxima página.

¹Vide [4, seção 1.1] para uma definição mais ampla desse serviço.

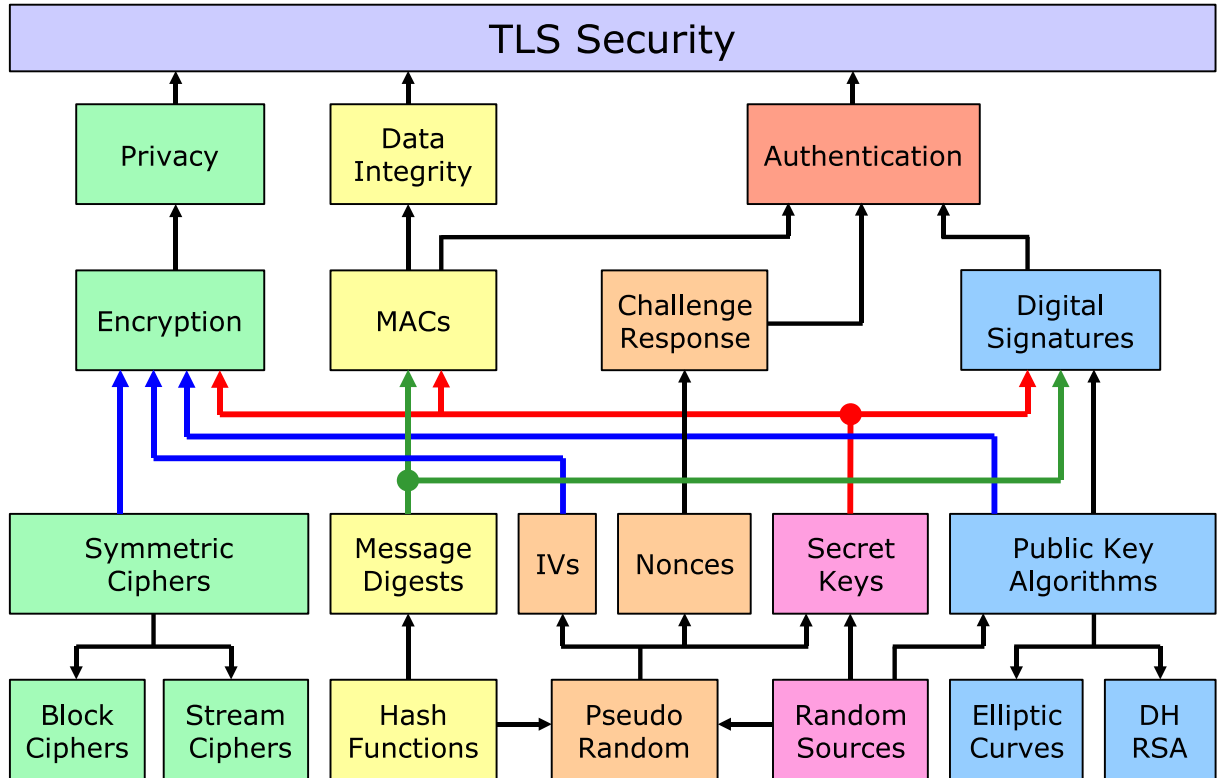





Figura 1: Ferramentas criptográficas e suas inter-correlações

Tabela 1: Simbologia usada nas definições dos algoritmos

Símbolo	Significado
m	Texto legível ou mensagem.
c	Texto cifrado.
h, i, r, s, v	Outros resultados/textos.
A	Entidade origem.
B	Entidade destino.
D, E, H, I, S, V	Funções/transformações.
$\delta, \varepsilon, \kappa$	Chaves criptográficas.
	Chave secreta (κ).
	Chave pública (δ) da entidade “E”.
	Chave privada (ε) da entidade “E”.

1.1 Funções de Hash (resumo criptográfico)

Uma Função de Hash H retorna, para uma mensagem m de tamanho arbitrário qualquer, uma sequência de bytes h de tamanho fixo, chamada valor de hash ou resumo criptográfico. Mais precisamente:

$$h = H(m)$$

Dentre as diversas funções de Hash existentes, o TLS limita-se aos algoritmos MD5 [5] e SHA-1 [6], cujos resumos são de 16 e 20 bytes respectivamente.

O valor de hash quando representa a mensagem da qual derivou é chamado de *message digest*.

1.2 Algoritmos de Chave Secreta

Os algoritmos de chave secreta (*Secret Key Algorithms* ou SKAs) se caracterizam pela utilização de uma mesma chave para as operações complementares (cifrar/decifrar ou assinar/verificar).

A segura utilização destes algoritmos demanda, entretanto, pela solução de três problemas intrínsecos:

1. A distribuição confidencial da chave secreta entre duas entidades que precisa ser efetuada *a priori* por (outro) canal seguro;
2. O crescimento exponencial, da ordem de $O(n^2)$ ², da quantidade de chaves em função do número de entidades envolvidas (n)³;
3. Como a sigilosidade de uma chave secreta decai na medida em que aumenta a sua exposição, torna-se imperativo portanto que esta chave secreta tenha um curto ciclo de vida, requisito este difícil de ser atendido, considerando-se os dois problemas citados acima.

No TLS, assim como em outros criptossistemas, esses problemas são resolvidos com o uso de algoritmos de chave pública (ver seção 1.3).

²A rigor: $\lim_{n \rightarrow \infty} \frac{n(n-1)}{2} = n^2$

³Assumindo que será utilizada uma chave única para cada par de entidades.

1.2.1 Message Authentication Code

O código de autenticação de mensagem (MAC), também chamado de *Message Integrity Code* (MIC), é uma especialidade de resumo criptográfico computado em função da mensagem m e de uma chave secreta κ . A autenticidade da origem e a integridade da mensagem podem ser comprovadas pelo destinatário usando-se a mesma chave κ , como ilustra a Figura 2, na qual o algoritmo de MAC/MIC está simbolizado pela função I e o resumo por i .

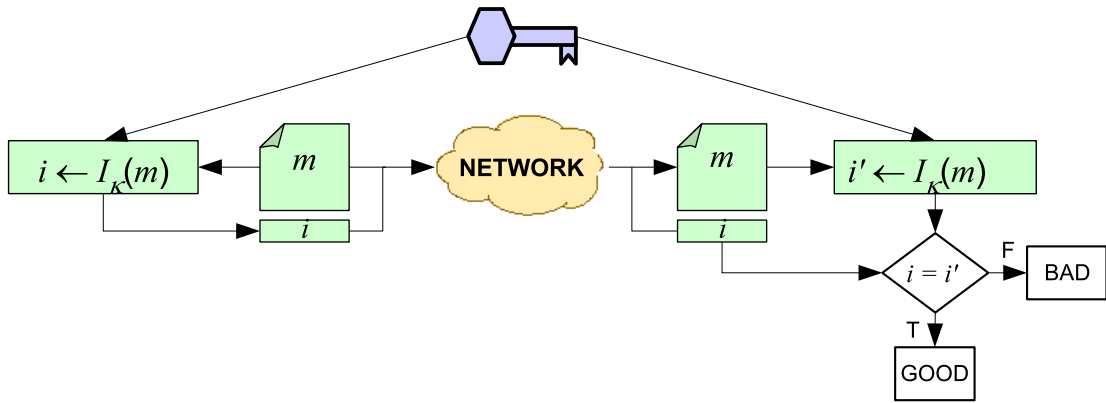


Figura 2: Geração e validação do MAC

Dentre os diversos tipos de MAC já elaborados, o único utilizado no TLS é o baseado em funções *Hash*, mais precisamente o *Hash-based Message Authentication Code* (HMAC) especificado em [7] e que pode ser resumido na seguinte equação:

$$i = \text{HMAC}_{\kappa}(H, m) = H(\kappa \oplus \text{opad} \parallel H(\kappa \oplus \text{ipad} \parallel m))$$

onde:

- $H(a)$ Uma função de *Hash* qualquer aplicada à sequência a . No caso do TLS, H está limitado a MD5 ou SHA-1.
- κ Chave secreta com 64 bytes completada com zeros à direita.
- \parallel Operador de concatenação de seqüências.
- \oplus Operador binário OU-Exclusivo.
- ipad O byte 0x36 repetido 64 vezes.
- opad O byte 0x5C repetido 64 vezes.

1.2.2 Cifradores Simétricos

Os cifradores simétricos utilizam a mesma chave κ para o processo de cifragem (E_{κ}) e decifragem ($D_{\kappa} \equiv E_{\kappa}^{-1}$), conforme ilustra a Figura 3 na próxima página.

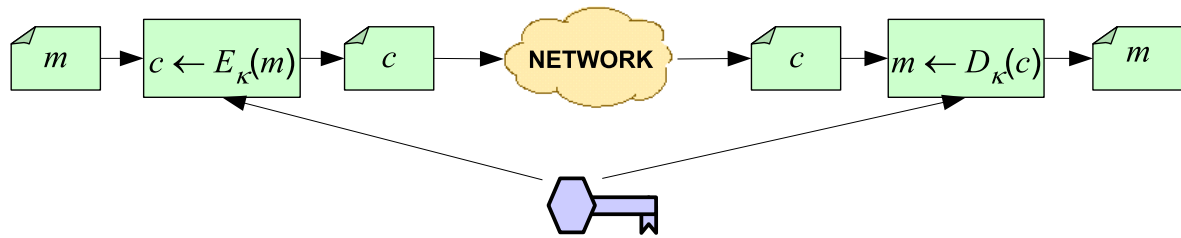


Figura 3: Cifragem e decifragem usando SKA

Cifradores simétricos são normalmente classificados em sequenciais (*stream ciphers*), que cifram/decifram um byte por vez, ou cifradores de bloco (*block ciphers*), que atuam sobre blocos de tamanho fixo, usualmente 8 ou 16 bytes.

Dentre os cifradores simétricos oficialmente suportados pelo TLS, o único sequencial é o RC4 [8]. Os demais cifradores (de bloco) relacionados na norma são DES [9], Triple-DES [10], RC2 [11] e IDEA [12], que usam blocos de 8 bytes.

1.3 Algoritmos de Chave Pública

Os algoritmos de chave pública (*Public Key Algorithms* ou PKAs) usam um par de chaves matematicamente complementares, sendo uma delas tornada pública (doravante denotada por δ) e a outra, chamada chave privada (simbolizada por ε), que deve ser conhecida apenas pela entidade proprietária.

Esses algoritmos apresentam uma solução simples e revolucionária para o problema de distribuição de chaves, já que as chaves públicas não precisam trafegar por canais confidenciais. No entanto, a autenticidade da origem das chaves e a integridade destas precisam estar asseguradas (ver seção 1.4).

Como é necessário distribuir apenas uma chave por entidade, os PKAs estão naturalmente isentos do segundo problema apresentado pelos SKAs, já que o crescimento da quantidade de chaves é linear.

Em função de sua maior complexidade computacional, os PKAs são tipicamente 10.000 vezes mais lentos que os SKAs. O TLS emprega então um criptossistema híbrido onde os SKAs são usados para os serviços de confidencialidade e integridade, com chaves secretas dinâmicas estabelecidas através de PKAs, que são também empregados para o serviço de autenticidade da origem e assinatura digital.

Os PKAs são tradicionalmente classificados conforme as seguintes propriedades [13]: cifragem, assinatura digital e estabelecimento de chave.

1.3.1 Cifragem

Quando é reversível, ou seja, as duas transformações (denotadas por E e D) são inversas entre si.

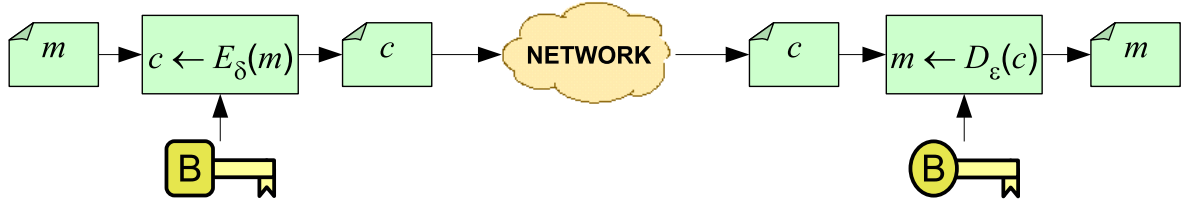


Figura 4: Cifragem e decifragem com PKA

O único algoritmo reversível oficializado na norma é o RSA [14].

1.3.2 Assinatura Digital

Quando a sua chave privada pode ser utilizada por uma transformação $S_ε$ para a geração de um resumo especial (a “assinatura”) a partir da texto legível. Este resumo será usado pela entidade destino que, com o uso da chave pública da entidade origem e da função complementar $V_δ$, poderá comprovar autenticidade e integridade da mensagem recebida.

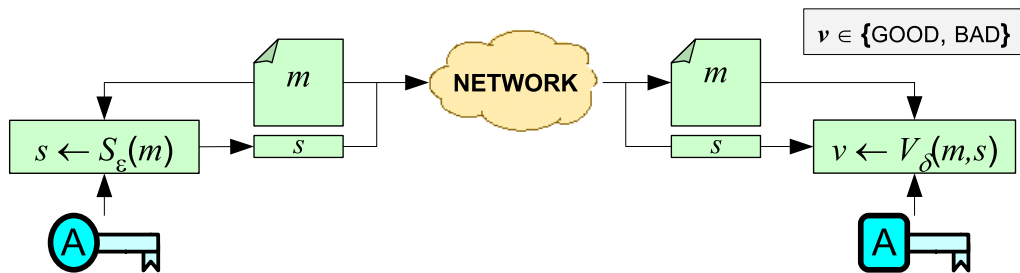


Figura 5: Assinatura Digital com PKA

A norma TLS especifica os algoritmos *Digital Signature Standard* (DSS) [15] e o RSA que, por ser reversível, pode ser usado aplicando-se as seguintes relações de equivalência:

$$\begin{aligned} S_ε(m) &\equiv E_ε(H(m)) \\ V_δ(m, s) &\equiv D_δ(s) \stackrel{?}{=} H(m) \end{aligned}$$

1.3.3 Estabelecimento de Chave

Quando o PKA pode ser usado para a criação de um protocolo que permite o compartilhamento de uma chave secreta entre as duas entidades participantes, mas ainda assim confidencial. Existem dois métodos para este fim:

Transferência – Uma das entidades é responsável pela geração da chave secreta que é enviada para a outra entidade após ser cifrada com chave pública desta, garantindo assim a confidencialidade da transferência. O PKA utilizado nesse caso deve ser obviamente reversível, como por exemplo o RSA.

Comum Acordo – As duas entidades contribuem conjuntamente para a geração da chave secreta utilizando somente parâmetros criptográficos públicos, incluindo as chaves públicas de ambas as entidades. O Diffie-Hellman (DH) [16] é um exemplo típico de algoritmo dessa categoria.

1.4 Certificados Digitais

No TLS todas as chaves públicas (PKs) devem ser armazenadas e distribuídas em certificados digitais segundo o formato ITU-T X.509 versão 3 [17]. A principal finalidade de um certificado é vincular uma chave pública à uma entidade final, conhecedora da respectiva chave privada, e identificada por um nome universalmente único, denominado *Distinguished Name* (DN).

A confiança na veracidade dessa associação é garantida pela assinatura digital emitida por uma entidade confiável *a priori*, chamada Autoridade Certificadora ou *Certification Authority* (CA).

Um certificado contém basicamente as seguintes informações:

- DN da entidade final;
- PK correspondente com todos os seus parâmetros criptográficos necessários, junto com uma identificação do algoritmo (RSA ou DSS);
- Período de validade;
- DN da CA emissora;
- Assinatura digital emitida pela CA.

Um certificado contém também a definição sobre a política de utilização da PK nele contida, se esta pode ou não ser utilizada para assinar outros certificados.

Quanto à política de utilização da sua PK e o tipo de assinatura do seu certificado, as entidades são normalmente classificadas em três categorias:

Tipo de Entidade	Uso da PK	Assinatura do seu certificado
Entidade final	Pode ser usada para qualquer propósito <u>exceto</u> para a assinatura digital.	Assinado por uma CA.
CA Intermediária	Somente para a assinatura digital de outros certificados	Assinado por outra CA.
CA Raiz	Idem	Auto-assinado.

A seqüência de certificados:

$$C_0, C_1, C_2, \dots, C_n$$

onde:

$$\begin{array}{ll} C_0 & \text{Certificado da CA raiz, ou seja, auto-assinado.} \\ C_1, \dots, C_{n-1} & \text{Certificados das CAs Intermediárias.} \\ C_n & \text{Certificado da entidade final.} \end{array}$$

é chamada caminho de certificação (“*Certification Path*”) da entidade final associada ao certificado C_n .

2

Introdução ao protocolo TLS

É apresentada a seguir uma descrição básica do protocolo TLS. Uma descrição completa do protocolo pode ser encontrada na especificação oficial [1] ou mais detalhadamente no livro “*SSL and TLS: Designing and Building Secure Systems*” [18].

2.1 O criptossistema TLS

O TLS é um criptossistema híbrido onde todos os parâmetros dos algoritmos simétricos são dinâmicos e derivados de um parâmetro-mestre chamado *master-key*, por sua vez definido através de um PKA para o estabelecimento de chave.

No TLS os algoritmos criptográficos não podem ser combinados livremente mas são pré-arranjados em forma de *ciphersuites*. Um *ciphersuite* é basicamente uma quádrupla $\langle Au, Kx, Enc, Mac \rangle$ que denota respectivamente os algoritmos de autenticação, de estabelecimento de chave, de cifragem e a função de *Hashing* usado no cômputo do HMAC.

2.2 Conexão e Sessão TLS

Uma conexão TLS é um canal transiente estabelecido entre duas aplicações tendo por base um protocolo de transporte confiável, tipicamente o TCP. Uma conexão tem fundamentalmente duas fases distintas: *handshake* e transferência de dados das aplicações (*bulk data transfer*).

O *handshake* TLS possui três propósitos:

1. Negociar o *ciphersuite* e os parâmetros da sessão TLS (definida abaixo);
2. Derivar todos os demais parâmetros criptográficos usados pelos algoritmos simétricos;

3. Autenticar as partes envolvidas (opcional).

Uma sessão TLS é caracterizada pelos seguintes atributos:

Version	Versão do protocolo TLS ou SSL.
Session ID	Identificador da sessão, definido pelo servidor.
master-key	Chave mestre definida através de um PKA para o estabelecimento de chave.
$\langle Enc, Mac \rangle$	Algoritmos de cifragem e MAC, subconjunto do <i>cipher-suite</i> .
Compression Method	Algoritmo de compressão.

Uma vez constituída, uma sessão pode ser restaurada nas conexões subseqüentes, economizando assim o alto custo computacional exigido para seu estabelecimento.

2.3 Arquitetura do protocolo TLS

O TLS é implementado de modo a atuar entre a camada de aplicação e a de transporte, como ilustra a Figura 6 adaptada de [19].

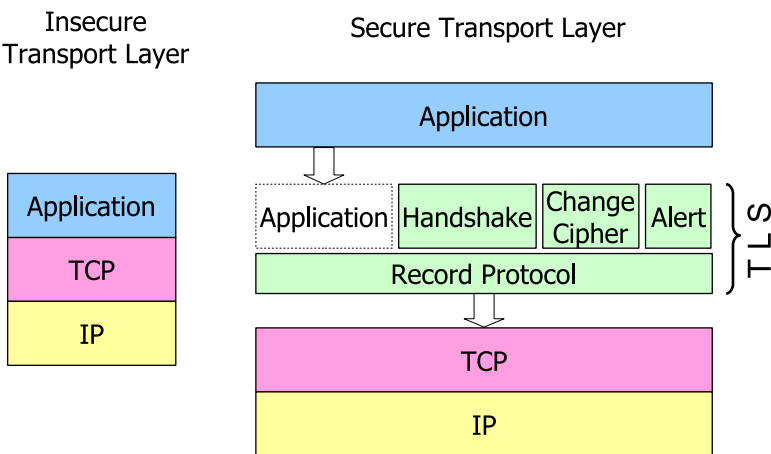


Figura 6: TLS na arquitetura em camadas de protocolos

O TLS pode ser decomposto em quatro sub-protocolos dispostos em duas camadas, sendo um inferior, chamado de *Record Protocol* e três superiores a saber: *Handshake*, *Alert* e *Change Cipher Spec (CCS)*.

Do ponto de vista funcional, o CCS, composto de uma única mensagem homônima, é parte integrante do *handshake* e assim será abordado neste texto. A sua discriminação

como um protocolo em separado serve apenas ao único propósito de garantir que a sua mensagem não compartilhe o mesmo registro com as demais mensagens do *handshake*.

2.3.1 TLS Record Protocol

Um registro TLS encapsula uma ou mais mensagens recebidas de um dos quatro protocolos superiores, e processadas conforme os mesmos parâmetros criptográficos correntes na conexão TLS.

Conforme exposto na Figura 7, toda mensagem oriunda das camadas superiores é primeiramente fragmentada em blocos de até 2^{14} bytes (16 kB), sendo eventualmente comprimida em seguida¹.

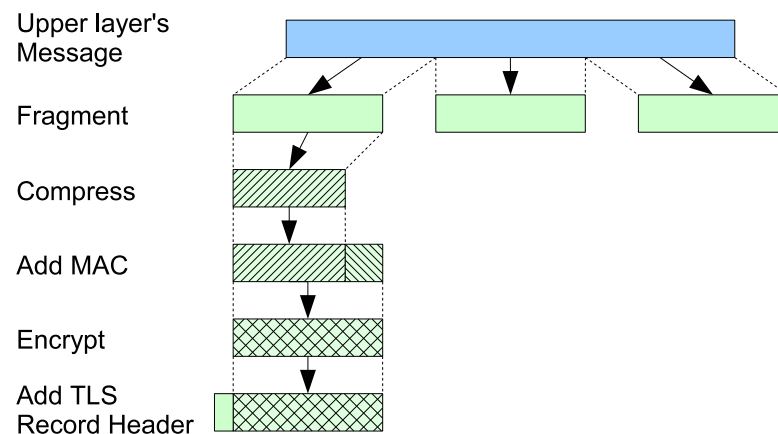


Figura 7: TLS Record Protocol

O próximo estágio no processamento da mensagem é o cálculo e o acréscimo do seu MAC, provendo assim a verificabilidade da integridade da mensagem e da autenticidade da sua origem.

Se o cifrador corrente for do tipo *block cipher*, acrescenta-se tantos bytes extras quanto forem necessários até que a soma do fragmento comprimido mais o MAC seja um múltiplo inteiro do tamanho do bloco do cifrador.

Finalmente ocorre a cifragem de toda a mensagem concatenada ao seu MAC usando um cifrador simétrico, sendo em seguida prefixada por um *header* de cinco bytes ilustrado na Figura 8 e descrito na Tabela 2 na próxima página.

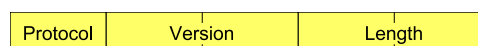


Figura 8: TLS Record Header

¹A RFC 2246 não especifica nenhum algoritmo de compressão.

Campo	Tamanho	Descrição										
<i>Protocol</i>	1	Indica o protocolo superior das mensagens encapsuladas segundo a seguinte codificação:										
		<table><tr><th>Tipo</th><th>Protocolo</th></tr><tr><td>20</td><td><i>Change Cipher Spec</i></td></tr><tr><td>21</td><td><i>Alert</i></td></tr><tr><td>22</td><td><i>Handshake</i></td></tr><tr><td>23</td><td><i>Application Data</i></td></tr></table>	Tipo	Protocolo	20	<i>Change Cipher Spec</i>	21	<i>Alert</i>	22	<i>Handshake</i>	23	<i>Application Data</i>
Tipo	Protocolo											
20	<i>Change Cipher Spec</i>											
21	<i>Alert</i>											
22	<i>Handshake</i>											
23	<i>Application Data</i>											
<i>Version</i>	2	Versão do protocolo. A versão 1.0 do TLS é codificada como 03 01 (em hexadecimal).										
<i>Length</i>	2	Tamanho total do conteúdo do registro, não podendo exceder a $2^{14} + 2048$.										

Tabela 2: TLS Record Header

O *ciphersuite* ativo no início de uma conexão é o $\langle NULL, NULL, NULL, NULL \rangle$.

2.3.2 *Alert Protocol*

A única mensagem que compõe esse protocolo pode ser enviada assincronamente por qualquer uma das partes para notificar erros para o outro par.

2.3.3 *Handshake Protocol*

O *handshake*, por ser o estágio certamente mais complexo, será analisado mais detalhadamente a seguir através da análise de três casos típicos.

Para evitar aspectos do protocolo TLS desnecessários tendo em vista o escopo deste projeto, o RSA será o PKA para o estabelecimento de chave empregado em todos os *handshakes* que se seguem.

2.3.3.1 *Handshake com autenticação do servidor*

A autenticação do servidor é realizada em praticamente todas as negociações TLS. A Tabela 3 na página 32 sumariza cada uma das mensagens exibidas na Figura 9.

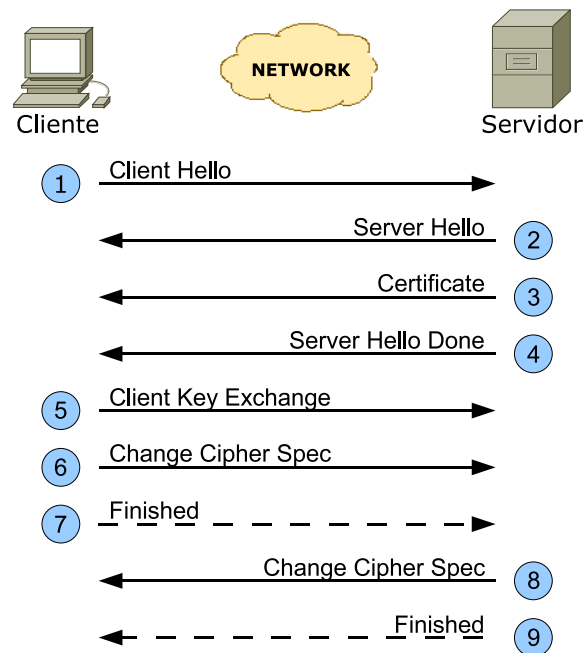


Figura 9: Handshake completo com autenticação do servidor

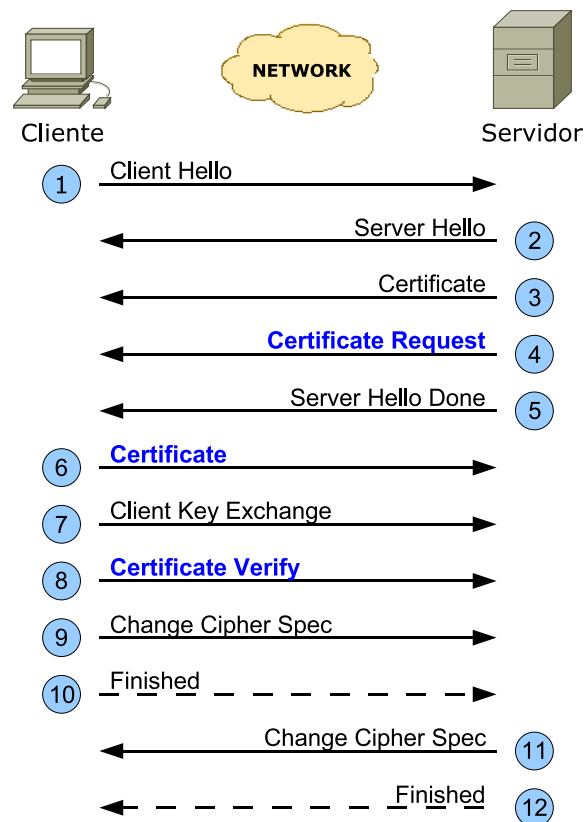


Figura 10: Handshake completo com autenticação mútua

Tabela 3: Handshake completo com autenticação do servidor

#	Descrição
1	O cliente envia a mensagem <i>Client Hello</i> propondo os seguintes parâmetros para a conexão: <ul style="list-style-type: none">• <i>Version</i> – Versão máxima do protocolo suportada (3.1 para o caso do TLS 1.0);• <i>Nonce</i> – Um número (pseudo-) aleatório de 32 bytes recém gerado para esta conexão;• <i>Session ID</i> – Identificador de uma sessão previamente estabelecida, caso o cliente deseje restaurá-la;• <i>Cipher Suites</i> – Lista dos parâmetros criptográficos suportados pelo cliente;• <i>Compression Methods</i> – Lista dos métodos de compressão suportados.
2	Servidor responde com <i>Server Hello</i> selecionando os parâmetros dentre aqueles propostos pelo cliente e incluindo o seu próprio <i>Nonce</i> .
3	Servidor envia seu certificado e todos os intermediários que porventura existam.
4	O servidor conclui esse primeiro estágio de negociação com a mensagem <i>Server Hello Done</i> .
5	O cliente gera uma chave de sessão aleatória e a envia cifrada com a chave pública do servidor na mensagem <i>Client Key Exchange</i> .
6	Através da mensagem <i>Change Cipher Spec</i> o cliente efetiva os parâmetros criptográficos já acordados. Todas as mensagens subsequentes passam a ser cifradas conforme esses novos parâmetros.
7	Um <i>hash</i> de todas as mensagens enviadas e recebidas é enviado para o servidor para evitar que negociação não sofra nenhuma interferência indevida.
8	O servidor também efetiva os parâmetros recém-negociados.
9	O servidor envia o <i>hash</i> calculado sobre as mensagens enviadas e recebidas para reforçar a integridade da negociação.

2.3.3.2 Handshake com autenticação mútua

A Tabela 4 resume as mensagens adicionais destacadas em negrito na Figura 10 na página 31.

Tabela 4: Handshake completo com autenticação mútua

#	Descrição
4	O servidor solicita ao cliente que envie o seu certificado X.509. A mensagem <i>Certificate Request</i> inclui uma lista das CAs aceitáveis pelo servidor.
6	O cliente envia o seu certificado e todos os eventuais certificados intermediários (não raiz).
8	Adicionalmente o cliente precisa enviar uma prova de autenticidade, comprovando que possui a chave privada correspondente à chave pública incluída no seu certificado.

2.3.3.3 Handshake abreviado

Esse mecanismo, denominado pela especificação de *Session Resumption*, permite a restauração de uma sessão TLS negociada anteriormente, reduzindo todo o *handshake* a apenas 6 breves mensagens e dispensando todo esforço computacional necessário para estabelecer uma nova conexão TLS. A Tabela 5 na próxima página descreve as principais mensagens que determinam a restauração de uma sessão, cujo processo completo encontra-se ilustrado na Figura 11.

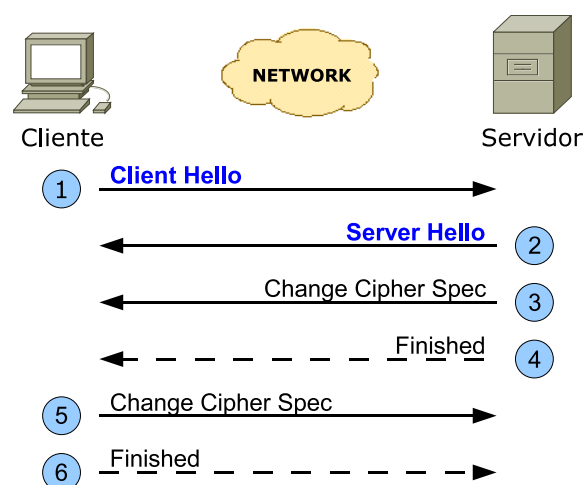


Figura 11: Handshake abreviado

Tabela 5: Handshake *abreviado*

#	Descrição
1	O cliente envia a mensagem <i>Client Hello</i> contendo todos os parâmetros necessários para o estabelecimento de uma nova sessão, mas inclui o identificador de sessão (“ <i>Session ID</i> ”) extraído da <i>Server Hello</i> enviada por este servidor em uma sessão anterior.
2	Servidor responde com <i>Server Hello</i> contendo o mesmo identificador presente na <i>Client Hello</i> , sinalizando que a sessão será restaurada.

3

As extensões do protocolo TLS

As recomendações presentes na RFC 3546 podem ser desmembradas em duas partes distintas. A primeira trata da especificação de um mecanismo genérico de codificação e negociação das extensões ao TLS. A segunda parte especifica 6 extensões que, em geral, visam reduzir o *overhead* do protocolo TLS.

As extensões codificadas devem ser acrescentadas ao final das mensagens *Client Hello* e *Server Hello*. Essa abordagem permite a compatibilidade com as implementações de TLS pré-existentes, uma vez que a sua especificação determina claramente que (RFC 2246, seção 7.4.1.2):

Forward compatibility note:

In the interests of forward compatibility, it is permitted for a client hello message to include extra data after the compression methods. This data must be included in the handshake hashes, but must otherwise be ignored. This is the only handshake message for which this is legal; for all other messages, the amount of data in the message must match the description of the message precisely.

As principais diretrizes na negociação das extensões são:

- Somente ao cliente é permitido propor extensões;
- O servidor pode, para cada extensão proposta, tomar uma das seguintes ações:
 1. Aceitá-la e sinalizar essa aceitação acrescentando a mesma extensão ao *Server Hello*;
 2. Ignorá-la silenciosamente;
 3. Rejeitá-la enviando uma mensagem *Alert* para o cliente e abortando o *handshake* imediatamente.
- Caso o servidor não confirme explicitamente a extensão proposta, o cliente pode abortar a conexão caso julgue a extensão imprescindível.

A segunda parte da RFC 3546 propõe seis extensões específicas que acrescentam novas funcionalidades ao protocolo TLS. Soma-se a elas uma sétima extensão não-oficial especificada nesse documento e também implementada no OpenSSL.

Apresenta-se a seguir uma descrição sistemática de cada uma dessas extensões, enfatizando os seguintes aspectos:

Referência - Indica o trecho de documentação que contém os detalhes técnicos sobre a extensão;

Sinopse - Um breve histórico da questão a ser resolvida e/ou aprimorada pela extensão;

Propósito da Extensão - Descreve a solução proposta pela extensão;

3.1 *Server Name Indication* (SNI)

Referência

[RFC 3546, seção 3.1.](#)

Sinopse

Servidores web, principalmente os comerciais, normalmente hospedam mais de um *site* através de um mecanismo chamado *name-based virtual hosting*, disponível em praticamente todos os servidores HTTP existentes. A seleção do domínio é feita pelo cliente HTTP (*browser*) através da campo ‘Host’ presente no *request* HTTP [20, seção 14.23].

Nos servidores HTTPS a seleção do domínio precisa ser efetuada precocemente, pois é decisiva para a seleção do certificado do servidor a ser enviado, podendo também impactar na escolha de outros parâmetros da sessão.

O campo ‘Host’ é, portanto, ineficiente para esta seleção uma vez que o *request* HTTP é transmitido tardiamente, após a conclusão do *handshake* TLS.

Propósito da Extensão

A solução natural oferecida nesta extensão consiste na inclusão do nome do *host* na primeira mensagem enviada pelo cliente, que é a *Client Hello*.

3.2 *Maximum Fragment Length* (MFL)

Referência

[RFC 3546, seção 3.2.](#)

Sinopse

Na camada *record layer*, cada registro recebido só pode ser disponibilizado para o protocolo superior após a verificação da sua integridade. Esse requisito obriga o cliente a dispor de memória suficiente para armazenar o maior registro admitido pela especificação que é cerca de 16 kB.

Propósito da Extensão

Essa extensão permite que o cliente estabeleça um limite superior para o tamanho de um fragmento TLS, podendo variar de 512 até 4096 bytes.

3.3 *Client Certificate URL (CCU)*

Referência

[RFC 3546, seção 3.3.](#)

Sinopse

Sempre que a autenticação do cliente é solicitada pelo servidor, o cliente deve enviar uma mensagem *Certificate* contendo um ou mais certificados.

Propósito da Extensão

Com o uso dessa extensão, a mensagem *Certificate* é substituída pela *Certificate URL* que conteria uma ou mais URLs para a obtenção de toda a cadeia de certificados do cliente.

3.4 *Trusted CA Indication (TCI)*

Referência

[RFC 3546, seção 3.4.](#)

Sinopse

A princípio, um servidor pode estar configurado com mais de um certificado para um mesmo *Common Name* emitidos por CAs diferentes.

Propósito da Extensão

Essa extensão serviria então para informar ao servidor quais são os CAs conhecidos e confiáveis pelo cliente. O servidor teria então um critério para efetuar uma escolha inequívoca sobre qual certificado X.509 deve empregado no *handshake* TLS.

3.5 *Truncated HMAC (TMAC)*

Referência

[RFC 3546, seção 3.5.](#)

Sinopse

O MAC acrescentado a cada mensagem cifrada TLS, que pode ter 16 ou 20 bytes de tamanho conforme o algoritmo de *Hash* em uso, pode representar uma parcela significativa nos dados trafegados, particularmente em comunicações interativas.

Propósito da Extensão

Essa extensão permite que tanto cliente quanto servidor passem a usar um HMAC reduzido (truncado) a 10 bytes.

3.6 *Certificate Status Request (CSR)*

Referência

[RFC 3546, seção 3.6.](#)

Sinopse

O protocolo *Online Certificate Status Protocol* (OCSP) [21] permite verificar em tempo real se um certificado foi revogado ou não pelo CA emissor.

Propósito da Extensão

Com o uso dessa extensão, o servidor atuaria como um *proxy* para a obtenção de uma resposta OCSP sobre o estado (revogado ou não-revogado) do certificado do próprio servidor, dispensando assim o estabelecimento de uma segunda conexão TCP com o servidor (“*responder*”) OCSP.

O uso do *proxy* seria benéfico também para os clientes que estivessem sujeitos a uma conectividade restrita, restrição comumente empregada em VPNs para redes corporativas.

3.7 *Server Certificate Chain Indication (SCCI)*

Referência

Draft em andamento.
Autor: Eliézio Oliveira

Sinopse

Como a cada novo *handshake* TLS (portanto, sem “*session resumption*”), o servidor envia quase sempre a mesma seqüência de certificados, que invariavelmente

corresponde a maior parte do volume de dados trafegados, decidimos acrescentar uma sétima extensão, a fim de evitar esse tráfego redundante.

Propósito da Extensão

Ao propor essa nova extensão, o cliente apresenta os identificadores de cada certificado presente na cadeia de certificados previamente obtida. Esses identificadores são exatamente os mesmos utilizados na extensão *Trusted CA Indication*.

Ao concordar com essa extensão, o servidor terá a opção de enviar uma mensagem *Certificate* vazia, mas ainda assim terá a alternativa de mandar a cadeia inteira. O cliente deve, portanto, estar preparado para tratar os dois casos ao processar a mensagem *Certificate* enviada pelo servidor.

Para eliminar qualquer possibilidade desta extensão deteriorar a segurança do protocolo TLS, optou-se por incluir toda a sequência de certificados no cômputo do resumo criptográfico a ser publicado na mensagem *Finished*. Essa operação deve ser efetuada imediatamente após a inclusão da mensagem *Certificate* recebida (cliente) ou enviada (servidor).

Implementação das extensões

4.1 Diretrizes de desenvolvimento

As extensões abordadas neste projeto foram implementadas como modificações no *software open-source* OpenSSL [22], versão 0.9.8a lançada em 11 de Outubro de 2005. Recomendamos o livro “*Network Security with OpenSSL*” [23] que contém uma extensa documentação da API do OpenSSL, bem como guias de utilização das diversas ferramentas que compõem o *toolkit*.

O OpenSSL assim modificado foi testado com sucesso nas plataformas Microsoft Windows 2000 Professional e Linux/x86, mas acredita-se que não haverá problemas de portabilidade para as outras dezenas de plataformas suportadas oficialmente pelo OpenSSL.

Optou-se por estender a API do OpenSSL com 15 funções assim agrupadas:

- 12 funções a serem usadas pelo cliente TLS para selecionar e configurar as extensões que serão propostas na mensagem *Client Hello*. Obviamente essas funções devem ser acionadas antes do início do *handshake* TLS para que sejam efetivas;
- 2 funções que se destinam a definir *callbacks* no lado servidor para processar as extensões *Server Name Indication* e *Client Certificate URL*;
- 1 função utilizável pelo servidor para indicar quais as extensões que serão reconhecidas e confirmadas.

Detalharemos a seguir cada uma das extensões implementadas, destacando os seguintes aspectos:

Especificação – Trecho relevante da especificação oficial para facilitar a comprovação da conformidade da implementação;

Divergências – Pontos em que a implementação se desvia da especificação;

API Estendida – Detalhamento das funções específicas para aquela extensão;

Implementação – Documentação das modificações realizadas no código-fonte do OpenSSL;

Testes de Conformidade – *Traces* de mensagens capturadas de testes reais que demonstram a eficácia da implementação e suas eventuais limitações.

Como um típico projeto *open-source*, o OpenSSL não dispõe de uma documentação sobre a sua arquitetura de *software*. Uma breve descrição de algumas poucas estruturas usadas internamente podem ser encontradas no capítulo “*Advanced Programming Topics*” do livro “*Network Security with OpenSSL*” supracitado.

4.2 Modificações Gerais

4.2.1 Build

Todas as modificações feitas nos arquivos da distribuição oficial do OpenSSL foram condicionadas à definição da macro `OPENSSL_NO_TLSEXT` que está indefinida por default. Para defini-la, e conseqüentemente desabilitar todas as extensões, basta incluir a opção `no-tlsex` na configuração do ambiente de *build* do OpenSSL. Exemplo: `./config no-tlsex` para as plataformas UNIX/Linux.

4.2.2 Estruturas de Dados

Foi criado um novo *include header* “*tlx.h*” que, além de conter os protótipos das funções da API estendida, define também a estrutura `TLS_EXTENSIONS` responsável por conter a representação das extensões a serem propostas e outras estruturas de dados necessárias para o processamento destas. Vide o Anexo A para a definição completa desta estrutura.

Três das principais estruturas nativas do OpenSSL foram incrementadas para suportar as extensões alvo desse projeto:

`ssl.h::SSL_CTX`, `SSL`

Inclusão do campo `tlx` (tipo: `TLS_EXTENSIONS`) contendo todas as estruturas necessárias durante o *handshake*.

`ssl.h::SSL_SESSION`, `ssl_asn1.c::SSL_SESSION_ASN1`

Incluídos 3 campos responsáveis pela representação de todas as informações que devem ser persistidas junto com a sessão TLS, a saber: `tlx_servername`, `tlx_max_fragment_length_id` e `tlx_truncated_hmac`.

ssl3.h::SSL3_STATE

Acrescentados os campos que representam os parâmetros das 2 únicas extensões que impactam a conexão TLS após o *handshake*: `tlx_max_plain_length` e `tlx_truncated_hmac`.

4.2.3 Funções Nativas

Segue abaixo um resumo de todas as modificações genéricas efetuadas no código base do OpenSSL. As demais modificações específicas estão documentadas em cada extensão mais adiante.

s3_clnt.c::ssl3_client_hello

Passou a chamar a função responsável pelo acréscimo das extensões propostas (`tlx_write_request`) ao final da construção da mensagem *Client Hello*.

s3_clnt.c::ssl3_get_server_hello

Chamada da nova função `tlx_read_response` para processar as respostas do servidor TLS às extensões propostas, caso não tenha ocorrido uma restauração da sessão.

s3_lib.c::ssl3_clear

Modificada para restabelecer os valores default dos campos `tlx_max_plain_length` e `tlx_truncated_hmac` da estrutura `s3` (do tipo `SSL3_STATE`), que são, respectivamente, `SSL3_RT_MAX_PLAIN_LENGTH_DEFAULT` e `FALSE`.

s3_srvr.c::ssl3_get_client_hello

Em caso de nova sessão, chama a `tlx_read_request` para o processamento das eventuais extensões.

s3_srvr.c::ssl3_send_server_hello

Passou a chamar a `tlx_write_response` caso alguma extensão tenha sido confirmada.

ssl_asn1.c::i2d_SSL_SESSION

Foi incrementada para cuidar da serialização em ASN.1 dos três parâmetros que devem ser persistidos: `tlx_servername`, `tlx_max_fragment_length_id` e `tlx_truncated_hmac`.

ssl_asn1.c::d2i_SSL_SESSION

Expandida para tratar da de-serialização dos novos atributos que devem ser persistidos na sessão (vide descrição da `i2d_SSL_SESSION`).

`ssl_lib.c::SSL_new`

Passou a duplicar as extensões que por ventura estejam pré-definidas no `SSL_CTX` matriz.

`ssl_lib.c::SSL_CTX_new`

Foi expandida para iniciar explicitamente as representações das extensões a serem negociadas presentes no novo campo `tlsex` da estrutura `SSL_CTX`.

`ssl_sess.c::SSL_SESSION_new`

Passou a iniciar explicitamente os novos parâmetros persistentes da estrutura `SSL_SESSION`.

`ssl_sess.c::SSL_SESSION_free`

Foi incrementada para liberar a memória eventualmente ocupada pelo novo atributo `tlsex_servername`.

`ssl_txt.c::SSL_SESSION_print`

Foi expandida para imprimir todos os parâmetros das extensões que estejam definidos.

4.2.4 Novas Funções

Sempre que possível, as tarefas mais complexas foram delegadas para funções implementadas no novo módulo “*tlsex_lib.c*” que implementa também todas as funções exportadas na API.

Duas funções internas se destacam dentre as implementadas nesse módulo: a `tlsex_write_request` e a `tlsex_read_request` que são, respectivamente, responsáveis pelo envio e recepção das extensões serializadas. A primeira é chamada no último estágio de geração da mensagem *Client Hello*, dentro da função `ssl3_client_hello`.

A segunda (`tlsex_read_request`) é deliberadamente chamada no início do processamento da mensagem *Client Hello* pela função `ssl3_get_client_hello`, para possibilitar que o objeto da conexão SSL seja reconfigurado em função do nome do *host* indicado.

4.3 Testes de Conformidade e Interoperabilidade

4.3.1 Aplicações de teste

Diversas aplicações acompanham o *toolkit* OpenSSL, sendo invocadas como subcomando do executável chamado `openssl`. Duas destas aplicações foram amplamente

modificadas para exercitar a API estendida e realizar testes de certificação da implementação das extensões.

Segue abaixo a documentação das principais modificações implementadas nas aplicações `s_server` e `s_client`.

4.3.1.1 `s_server`

Novas opções:

`-tlsex_allow arg`

Determina as extensões que serão aceitas e confirmadas pelo servidor TLS. Atualmente `arg` deve ser sempre igual a `all`.

4.3.1.2 `s_client`

Todas as novas opções iniciadas com o prefixo `tlsex_` só podem ser usadas quando TLS é o único protocolo selecionado, o que implica no uso conjunto com a opção `-tls1`.

`-reconnect arg`

Essa opção já existia mas passou a aceitar um parâmetro opcional `arg` que determina o número de reconexões (default: 5).

`-reinit arg`

Semelhante a `-reconnect` mas impede que haja uma restauração da sessão nas conexões subseqüentes. `arg` é opcional e seu valor default é 5.

`-tlsex_server_name`

Habilita a extensão *Server Name Indication*. O nome de *host* é definido implicitamente pelo valor passado para a opção `-connect`.

`-tlsex_max_fragment_length length`

Habilita a extensão *Maximum Fragment Length* com o tamanho indicado.

`-tlsex_cert_url url!hash`

`-tlsex_pkipath_url url!hash`

Essas opções, que são mutuamente excludentes, habilitam e configuram a extensão *Client Certificate URL*. O *hash* associado ao objeto apontado pela URL é opcional e deve estar em notação hexadecimal e separado desta pelo sinal '!'. A opção `-tlsex_cert_url` pode ser empregada múltiplas vezes para definir uma cadeia de certificados elemento a elemento.

-tlsx_trusted_ca_id arg

Habilita a extensão *Trusted CA Indication* conforme o tipo indicado por *arg*, que deve assumir um dos seguintes valores: *pre_agreed*, *key_sha1_hash*, *x509_name* ou *cert_sha1_hash*.

-tlsx_truncated_hmac

Habilita a extensão *Truncated HMAC*.

-tlsx_server_cert_id arg

Habilita a extensão *Server Certificate Chain Indication*, com *arg* definindo o tipo de identificador a ser utilizado, podendo assumir os mesmos valores que o parâmetro da opção *-tlsx_trusted_ca_id*.

Uma vez estabelecida a conexão TLS, o aplicativo *s_client* entra em modo comando, enviando o que for digitado (*line buffered*) para o servidor, com os seguintes tratamentos especiais segundo a primeira letra do comando:

Tabela 6: Comandos especiais do aplicativo *s_client*

Letra	Ação
Q	Causa o fechamento das conexões TLS e TCP e o encerramento do programa.
N	(nova opção) Força uma reconexão TLS, possivelmente com restauração dos parâmetros da sessão (<i>“session resumption”</i>).
n	(nova opção) Força uma reconexão TLS após invalidar a sessão e com isso evitar que seja restaurada.

4.3.2 Ambiente de teste

O OpenSSL estendido foi compilado e executado nos sistemas operacionais Linux/i386 (distribuição Ubuntu 5.10, *kernel* 2.6.12) e Windows 2000 Professional, ambos sobre um *hardware* PC/Intel-Pentium. O computador utilizado está registrado no DNS com o nome dinâmico *eliezio.no-ip.info*.

O tráfego TLS foi capturado com o uso do aplicativo *tethereal* [24] versão 0.10.12, que já possui suporte parcial às extensões oficiais do TLS.

Listados a seguir encontram-se os *scripts* usados na ativação das aplicações de teste *s_client* e *s_server*:

```
#!/bin/sh
```

```
./apps/openssl s_server -accept 443 -cert server.crt -key server.key \  
-no_dhe -no_ecdhe -tls1 -CApath . -WWW -tlsex_allow all $*
```

Listagem 1: Script *s_server.sh*

```
#!/bin/sh
```

```
./apps/openssl s_client -connect eliezio.no-ip.info:443 \  
-cipher RC4-SHA -tls1 -CApath . $*
```

Listagem 2: Script *s_client.sh*

4.3.3 Testes de Interoperabilidade

Na pesquisa efetuada no início do projeto só foram identificadas duas outras implementações da RFC 3546, ambas parciais.

A pilha SSL/TLS GnuTLS [25] versão 1.0.14 implementa as extensões *Server Name Indication* e *Maximum Fragment Length*.

O navegador *web* Opera 9.0 [26], por sua vez, traz implementado as extensões *Server Name Indication* e *Certificate Status Request*.

4.3.4 Apresentação dos resultados dos testes

Os resultados dos testes comprovando a conformidade da implementação serão apresentados em geral em forma de listagens contendo um trecho da saída do decodificador *tethereal*, ressaltando as mensagens TLS impactadas pela extensão.

Em alguns casos específicos, onde uma visão mais ampla de toda a comunicação se fizer necessária, apresentaremos uma tabela sinóptica elaborada manualmente a partir da listagem dos pacotes TCP exibida pelo *ethereal*. Precisou-se recorrer à decodificação manual dos registros TLS devido à incapacidade do *ethereal* de decodificar corretamente registros fracionados em mais de um segmento TCP.

Time (s)	Direction C \longleftrightarrow S	TCP		TLS Record [length]
		Flags	Length	
0.000000	\longrightarrow	SYN		
0.028681	\longleftarrow	SYN, ACK		
0.028835	\longrightarrow	ACK		
0.031536	\longrightarrow		57	Client Hello [52]
0.072442	\longleftarrow		86	Server Hello [81]
0.072600	\longrightarrow	ACK		
0.096442	\longleftarrow		1452	Certificate R1 [1024], Certificate R2a [418/603]
...				

Nessas tabelas (vide exemplo na página precedente), os pacotes de uma conexão estão dispostos horizontalmente, um por fileira. As informações de cada pacote discriminadas em cada coluna são as seguintes:

<i>Time (s)</i>	Tempo transcorrido desde o início da captura, em segundos. Obtido diretamente do <i>ethereal</i> .
<i>Direction</i>	Direção do fluxo do pacote: cliente para servidor (\longrightarrow) ou servidor para cliente (\longleftarrow), que também possuem um sombreamento diferenciado. Inferido com base nos endereços IP origem e destino do <i>header IP</i> .
<i>TCP Flags</i>	Flags presentes no cabeçalho do pacote TCP, exibido quando este não contém nenhum dado. Exibido diretamente pelo <i>ethereal</i> e indicado pelo campo <i>Len=0</i> .
<i>TCP Length</i>	Tamanho da área de dados (<i>payload</i>) do segmento TCP. Indicado pelo campo <i>Len=n</i> , $n > 0$.
<i>TLS Record [length]</i>	Registros presentes no segmento TCP, com o tamanho total do registro indicado entre '[]'. Quando o registro está parcialmente representado no segmento, esse valor é apresentado na forma [<i>tamanho parcial/tamanho total</i>]. Quando uma mesma mensagem TLS é particionada em diversos registros, acrescenta-se a notação <i>Rn</i> para enumerar cada uma das partes. Caso um segmento TCP só contenha parte de um registro TLS, acrescentar-se-á uma letra a esta notação para enumerar estas partes.

4.4 *Server Name Indication*

4.4.1 Especificação

The "extension_data" field of this extension SHALL contain "ServerNameList" where:

```
struct {
    ServerName server_name_list<1..2^16-1>
} ServerNameList;

struct {
    NameType name_type;
    select (name_type) {
        case host_name: HostName;
    } name;
} ServerName;

enum {
    host_name(0), (255)
} NameType;

opaque HostName<1..2^16-1>;
```

Listagem 3: RFC 3546, trecho da seção 3.1

4.4.2 Divergências

Apesar da especificação permitir a definição de múltiplos nomes, a implementação restringe a indicação de um único nome, pois se julgou desnecessária aquela flexibilidade.

4.4.3 API Estendida

```
/* ----- Client Side ----- */
void TLSX_set_server_name (
    SSL * ssl,
    const char *server_name
);

/* ----- Server Side ----- */
void TLSX_CTX_set_servername_callback (
    SSL_CTX *ctx,
    int (*cb) (
```

```
    SSL *ssl,  
    const char *name,  
    void *uarg  
) ,  
    void *uarg  
);
```

Listagem 4: API para a extensão SNI

A função de *callback* `cb` definida pela `TLSX_CTX_set_servername_callback` será acionada logo após o recebimento da mensagem *Client Hello* contendo a extensão *Server Name Indication* ou quando restauração de uma antiga sessão (“*session resumption*”).

4.4.4 Implementação

`s3_srvr.c::ssl3_get_client_hello`

Em caso de restauração de sessão aciona a *callback* para tratar o nome do *host*, caso esteja definida.

4.4.5 Testes de Conformidade

A aplicação `s_server` foi modificada para configurar (através da `TLSX_CTX_set_servername_callback`) a função `ssl_servername_cb` como *callback* para o processamento dessa extensão.

A prova de conformidade será, portanto, a devida ativação desta função na aplicação `s_server` e a exibição da mensagem informando o *hostname* recebido na extensão.

```
static int MS_CALLBACK  
ssl_servername_cb (SSL *s, const char *servername, void *uarg)  
{  
    BIO_printf(bio_err, "Hostname in TLS extension: \"%s\"\n",  
               servername);  
  
    return SSL_ERROR_NONE;  
}
```

Listagem 5: Callback para tratar extensão SNI

```
./s_client -tlsx_server_name -reconnect 1
```

Listagem 6: Chamada do script de teste para a extensão SNI

4.4.5.1 Primeira Conexão (*full handshake*)

Secure Socket Layer

SSL Record Layer: Handshake Protocol: Client Hello

Content Type: Handshake (22)

Version: TLS 1.0 (0x0301)

Length: 74

Handshake Protocol: Client Hello

Handshake Type: Client Hello (1)

Length: 70

Version: TLS 1.0 (0x0301)

Random.gmt_unix_time: Jan 4, 2006 19:24:49.000000000

Random.bytes

Session ID Length: 0

Cipher Suites Length: 2

Cipher Suites (1 suite)

Cipher Suite: TLS_RSA_WITH_RC4_128_SHA (0x0005)

Compression Methods Length: 1

Compression Methods (1 method)

Compression Method: null (0)

Extensions Length: 27

Extension: server_name

Type: server_name (0x0000)

Length: 23

Data (23 bytes)

```

0000  00 00 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00  .....E.
0010  00 83 26 3e 40 00 40 06 6d fc c9 2d 09 f0 c9 2d  ..&>@.@.m...-
0020  09 f0 04 92 11 51 19 45 9b bc 19 42 9b 10 80 18  ....Q.E...B....
0030  20 00 a6 b0 00 00 01 01 08 0a 00 3a bd 97 00 3a  .....:.....:
0040  bd 81 16 03 01 00 4a 01 00 00 46 03 01 43 bc 3d  ....J...F..C.=
0050  21 af b8 32 9f 5e 4a 6b 30 7d f6 fe c1 d4 ff 4e  !..2.^Jk0}.....N
0060  5f c8 44 f1 74 bc df a5 1e be da af 6b 00 00 02  _..D.t.....k...
0070  00 05 01 00 00 1b 00 00 00 17 00 15 00 00 12 65  .....e
0080  6c 69 65 7a 69 6f 2e 6e 6f 2d 69 70 2e 69 6e 66  liezio.no-ip.inf
0090  6f  o

```

Listagem 7: Primeira Conexão – Decodificação da mensagem Client Hello

ACCEPT

Hostname in TLS extension: "eliezio.no-ip.info"

Shared ciphers:RC4-SHA

CIPHER is RC4-SHA

Listagem 8: Primeira Conexão – Saída da aplicação s_server

4.4.5.2 Segunda Conexão (*resumed handshake*)

Secure Socket Layer

SSL Record Layer: Handshake Protocol: Client Hello

Content Type: Handshake (22)

Version: TLS 1.0 (0x0301)

Length: 106

Handshake Protocol: Client Hello

Handshake Type: Client Hello (1)

Length: 102

Version: TLS 1.0 (0x0301)

Random.gmt_unix_time: Jan 4, 2006 19:24:50.000000000

Random.bytes

Session ID Length: 32

Session ID (32 bytes)

Cipher Suites Length: 2

Cipher Suites (1 suite)

Cipher Suite: TLS_RSA_WITH_RC4_128_SHA (0x0005)

Compression Methods Length: 1

Compression Methods (1 method)

Compression Method: null (0)

Extensions Length: 27

Extension: server_name

Type: server_name (0x0000)

Length: 23

Data (23 bytes)

```

0000  00 00 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00  .....E.
0010  00 a3 c9 ff 40 00 40 06 ca 1a c9 2d 09 f0 c9 2d  ....@.@.....-...-
0020  09 f0 04 93 11 51 18 ef 09 42 18 f9 cb 1b 80 18  ....Q...B.....
0030  20 00 a6 d0 00 00 01 01 08 0a 00 3a c0 c0 00 3a  ....:....:
0040  c0 ba 16 03 01 00 6a 01 00 00 66 03 01 43 bc 3d  ....j...f..C.=
0050  22 07 30 57 65 ff f3 0c 0b f7 26 02 2c 34 4c 80  ".0We.....&.,4L.
0060  7f 4a 8b 47 6e c4 df e8 fc d1 64 28 dc 20 d2 f6  .J.Gn.....d(. ..
0070  98 0e 3f 3e ea 7f 35 09 d3 79 f6 20 b0 93 d6 aa  ..?>..5..y. ....
0080  f1 b8 b5 90 ce 26 51 24 52 71 63 d2 53 dc 00 02  ....&Q$Rqc.S...
0090  00 05 01 00 00 1b 00 00 00 17 00 15 00 00 12 65  .........e
00a0  6c 69 65 7a 69 6f 2e 6e 6f 2d 69 70 2e 69 6e 66  liezio.no-ip.inf
00b0  6f  o

```

Listagem 9: Segunda Conexão – Decodificação da mensagem Client Hello

ACCEPT

Hostname in TLS extension: "eliezio.no-ip.info"

Shared ciphers:RC4-SHA

CIPHER is RC4-SHA
 Reused session-id

Listagem 10: Segunda Conexão – Saída da aplicação s_server

4.4.6 Testes de Interoperabilidade

Opera Browser 9.0

Secure Socket Layer

```
SSL Record Layer: Handshake Protocol: Client Hello
Content Type: Handshake (22)
Version: TLS 1.0 (0x0301)
Length: 130
Handshake Protocol: Client Hello
  Handshake Type: Client Hello (1)
  Length: 126
  Version: TLS 1.0 (0x0301)
  Random.gmt_unix_time: Mar 13, 2006 02:07:39.000000000
  Random.bytes
  Session ID Length: 0
  Cipher Suites Length: 14
  Cipher Suites (7 suites)
    Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
    Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
    Cipher Suite: TLS_RSA_WITH_RC4_128_SHA (0x0005)
    Cipher Suite: TLS_RSA_WITH_RC4_128_MD5 (0x0004)
    Cipher Suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x000a)
    Cipher Suite: TLS_RSA_WITH_DES_CBC_SHA (0x0009)
    Cipher Suite: TLS_RSA_EXPORT1024_WITH_RC4_56_SHA (0x0064)
  Compression Methods Length: 1
  Compression Methods (1 method)
    Compression Method: null (0)
  Extensions Length: 71
  Extension: server_name
    Type: server_name (0x0000)
    Length: 23
    Data (23 bytes)
  Extension: status_request
    Type: status_request (0x0005)
    Length: 40
    Data (40 bytes)
```

```
0000  00 c0 49 43 0b 55 00 0b db e1 13 ef 08 00 45 00  ..IC.U.....E.
0010  00 af e4 07 40 00 80 06 36 73 0a 04 02 a0 c9 2d  ....@...6s.....-
```

```

0020 09 fd 0f 7b 01 bb cc 44 56 f0 8d 02 f6 ff 50 18    ...{...DV.....P.
0030 44 70 04 b2 00 00 16 03 01 00 82 01 00 00 7e 03    Dp.....~.
0040 01 44 14 fe 1b 7e cf d8 90 df 56 2a c9 39 04 59    .D...~....V*.9.Y
0050 a6 03 60 f1 43 00 d7 a1 74 0e f1 bb 52 b1 c6 d6    ..'.C...t...R...
0060 e1 00 00 0e 00 35 00 2f 00 05 00 04 00 0a 00 09    .....5./.....
0070 00 64 01 00 00 47 00 00 00 17 00 15 00 00 12 65    .d...G.....e
0080 6c 69 65 7a 69 6f 2e 6e 6f 2d 69 70 2e 69 6e 66    liezio.no-ip.inf
0090 6f 00 05 00 28 01 00 00 00 23 22 21 30 1f 06 09    o...(....#"!0...
00a0 2b 06 01 05 05 07 30 01 02 04 12 04 10 25 95 4c    +.....0.....%.L
00b0 c1 d3 a5 37 c6 27 a0 bc 37 78 86 62 f0          ...7.'...7x.b.

```

Listagem 11: Conexão oriunda do Opera – Decodificação da mensagem Client Hello

Secure Socket Layer

TLS Record Layer: Handshake Protocol: Server Hello

Content Type: Handshake (22)

Version: TLS 1.0 (0x0301)

Length: 80

Handshake Protocol: Server Hello

Handshake Type: Server Hello (2)

Length: 76

Version: TLS 1.0 (0x0301)

Random.gmt_unix_time: Mar 13, 2006 02:04:09.000000000

Random.bytes

Session ID Length: 32

Session ID (32 bytes)

Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)

Compression Method: null (0)

Extensions Length: 4

Extension: server_name

Type: server_name (0x0000)

Length: 0

Data (0 bytes)

```

0000 00 0b db e1 13 ef 00 c0 49 43 0b 55 08 00 45 00    .....IC.U..E.
0010 05 d4 46 e5 40 00 74 06 da 70 c9 2d 09 fd 0a 04    ..F.@.t..p.-....
0020 02 a0 01 bb 0f 7b 8d 02 f6 ff cc 44 57 77 50 10    .....{.....DWwP.
0030 43 89 41 e9 00 00 16 03 01 00 50 02 00 00 4c 03    C.A.....P...L.
0040 01 44 14 fd 49 db 10 46 6f c1 21 ca 36 05 94 d3    .D..I..Fo.!.6...
0050 09 82 40 0a 4f f8 72 9b 21 b6 4d d0 ca 81 ba d2    ..@.0.r.!.M.....
0060 6f 20 b4 25 99 27 d2 ed ed ca 02 a5 42 d6 d7 c9    o .%. '.....B...
0070 a6 cb f0 12 f1 aa 13 77 8f 44 fa f0 e1 aa 0c ca    .....w.D.....
0080 b0 32 00 35 00 00 04 00 00 00 16 03 01 06 5b    .2.5.....[

```

Listagem 12: Resposta ao request do Opera 9 – Decodificação da mensagem Server Hello

4.5 Maximum Fragment Length

4.5.1 Especificação

The "extension_data" field of this extension SHALL contain:

```
enum{
    2^9(1), 2^10(2), 2^11(3), 2^12(4), (255)
} MaxFragmentLength;
```

whose value is the desired maximum fragment length. The allowed values for this field are: 2^9 , 2^{10} , 2^{11} , and 2^{12} .

Listagem 13: RFC 3546, trecho da seção 3.2

4.5.2 Divergências

O valor 5 ($2^{13} = 8192$) é também aceito.

4.5.3 API Estendida

```
/* ----- Client Side ----- */
void TLSX_CTX_set_maximum_fragment_length (
    SSL_CTX *ctx,
    int frag_length
);

void TLSX_set_maximum_fragment_length (
    SSL *ssl,
    int frag_length
);
```

Listagem 14: API para a extensão MFL

Em ambas as funções o valor efetivamente selecionado é menor tamanho válido segundo a especificação (512, 1024, 2048, 4096 ou 8192) que seja maior ou igual a frag_length.

4.5.4 Implementação

s3_srvr.c::ssl3_get_client_hello

Modificada para efetivar o parâmetro tlsx_max_plain_length em caso de restauração de sessão.

```
s3_clnt.c::ssl3_get_server_hello
```

Após a restauração de uma sessão TLS (*“session resumption”*), efetiva o parâmetro `tlx_max_plain_length` imediatamente;

4.5.5 Testes de Conformidade

A validação da implementação dessa extensão é um pouco mais complexa, pois a ferramenta de captura e decodificação utilizada (*tethereal*) não é capaz de tratar corretamente mensagens de *handshake* desmembradas em mais de um registro TLS. Resta-nos, portanto, efetuar uma captura menos detalhada que explicita principalmente o tamanho de cada registro TLS e assim certificar-nos de que o tamanho de cada registro não excede ao *maximum fragment length* especificado, e se esta decomposição não interfere no perfeito fechamento do *handshake*.

A captura diferenciada foi realizada usando-se a seguinte sintaxe na chamada do *tethereal*:

```
tethereal -i eth0 \
-z "proto,colinfo,ssl.record.length,ssl.record.length" \
-z "proto,colinfo,tcp.len>0,tcp.len" \
port 443
```

Listagem 15: Chamada do *tethereal* para o detalhamento dos registros TLS

O caso de teste escolhido consistiu em uma solicitação HTTPS usando o método GET do arquivo README da distribuição do OpenSSL que contém exatos 7930 bytes. A chamada do *script* `s_client.sh` ficou, portanto:

```
./s_client.sh -tlx_max_fragment_length 1024 -reconnect 1
```

Listagem 16: Chamada do `s_client.sh` para o teste da extensão MFL

```
Secure Socket Layer
SSL Record Layer: Handshake Protocol: Client Hello
Content Type: Handshake (22)
Version: TLS 1.0 (0x0301)
Length: 52
Handshake Protocol: Client Hello
Handshake Type: Client Hello (1)
Length: 48
Version: TLS 1.0 (0x0301)
Random.gmt_unix_time: Jan 6, 2006 11:27:27.000000000
Random.bytes
Session ID Length: 0
Cipher Suites Length: 2
Cipher Suites (1 suite)
```



```
Cipher Suite: TLS_RSA_WITH_RC4_128_SHA (0x0005)
  Compression Methods Length: 1
  Compression Methods (1 method)
Compression Method: null (0)
  Extensions Length: 5
  Extension: max_fragment_length
Type: max_fragment_length (0x0001)
Length: 1
Data (1 byte)
```

```
0000  00 c0 49 43 0b 55 00 50 04 aa 26 05 08 00 45 00  ..IC.U.P..&...E.
0010  00 61 67 39 40 00 40 06 f4 1b 0a 04 02 14 c9 2d  .ag9@.@.....-
0020  09 fd e7 86 01 bb d6 f1 40 b0 30 5d 60 1a 50 18  .....@.0]' .P.
0030  16 d0 54 f9 00 00 16 03 01 00 34 01 00 00 30 03  ..T.....4...0.
0040  01 43 be 70 3f b9 60 f0 1c 02 1b 8f e7 35 db 4f  .C.p?..'.....5.0
0050  c8 da 03 81 fc 8e 66 fb 9a 9b f8 01 68 08 79 16  .....f.....h.y.
0060  57 00 00 02 00 05 01 00 00 05 00 01 00 01 02  W.....
```

Listagem 17: Mensagem Client Hello com a extensão MFL habilitada

As duas conexões foram posteriormente analisadas e sintetizadas nas tabelas apresentas a seguir.

4.5.5.1 Primeira conexão (*full handshake*)

Pelo tráfego da primeira conexão, sintetizado na Tabela 7 na página seguinte, pode-se constatar a observância do limite imposto pelo *maximum fragment length* pela presença dos diversos registros TLS para a composição da mensagem *Certificate* enviada pelo servidor.

A Figura 12 ilustra como a mensagem *Certificate* é fragmentada em dois registros após a ativação dessa extensão, e como esses registros se distribuem entre dois segmentos TCP.

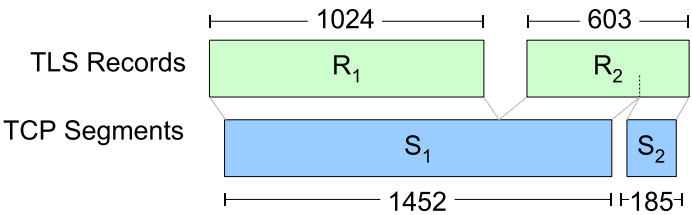


Figura 12: Fragmentação da mensagem *Certificate*

Tabela 7: Handshake com registros limitados a 1024 bytes

Time (s)	Direction C \longleftrightarrow S	TCP		TLS Record [length]
		Flags	Length	
0.000000	\longrightarrow	SYN		
0.028681	\longleftarrow	SYN, ACK		
0.028835	\longrightarrow	ACK		
0.031536	\longrightarrow		57	Client Hello [52]
0.072442	\longleftarrow		86	Server Hello [81]
0.072600	\longrightarrow	ACK		
0.096442	\longleftarrow		1452	Certificate R1 [1024], Certificate R2a [418/603]
0.096567	\longrightarrow	ACK		
0.096637	\longleftarrow		185	Certificate R2b [185/603]
0.096708	\longrightarrow	ACK		
0.143681	\longleftarrow		9	Server Hello Done [4]
0.143837	\longrightarrow	ACK		
0.154354	\longrightarrow		186	Client Key Exchange [134], Change Cipher Spec [1], Finished [36]
0.328340	\longleftarrow	ACK		
0.423803	\longleftarrow		6	Change Cipher Spec [1]
0.466072	\longrightarrow	ACK		
0.496468	\longleftarrow		41	Finished [36]
0.497304	\longrightarrow	ACK		
0.500684	\longrightarrow		27	Alert - Close Notify [22]
0.500964	\longrightarrow	FIN, ACK		
0.532371	\longleftarrow	ACK		
0.548267	\longleftarrow	FIN, ACK		
0.548366	\longrightarrow	ACK		

4.5.5.2 Segunda conexão (*resumed handshake*)

A segunda conexão realizada logo em seguida e cujos resultados estão sintetizados na Tabela 8 na página seguinte, comprova duas outras funcionalidades:

1. Que após a restauração da sessão o limite do tamanho de cada registro voltou a ser o valor negociado na sessão anterior (1024); e
2. Esse limite é efetivo após o *handshake*, como fica evidente pelo desmembramento da resposta (*Application Data*) em 8 registros.

Tabela 8: *Conexão TLS com registros limitados a 1024 bytes*

Time (s)	Direction C \longleftrightarrow S	TCP		TLS Record [length]
		Flags	Length	
0.501275	\longrightarrow	SYN		
0.533568	\longleftarrow	SYN, ACK		
0.533708	\longrightarrow	ACK		
0.534627	\longrightarrow		89	Client Hello [84]
0.658061	\longleftarrow		79	Server Hello [74]
0.659113	\longrightarrow	ACK		
0.691062	\longleftarrow		47	Change Cipher Spec [1], Finished [36]
0.696049	\longrightarrow	ACK		
0.697457	\longrightarrow		47	Change Cipher Spec [1], Finished [36]
0.926678	\longleftarrow	ACK		
8.144505	\longrightarrow		46	Application Data [41]
8.260117	\longleftarrow		1049	Application Data R1 [1044]
8.286967	\longleftarrow		1452	Application Data R2 [1044], Application Data R3a [398/1044]
8.287067	\longrightarrow	ACK		
8.291990	\longleftarrow		646	Application Data R3b [646/1044]
8.308943	\longleftarrow		1452	Application Data R4 [1044], Application Data R5a [398/1044]
8.309037	\longrightarrow	ACK		
8.314126	\longleftarrow		646	Application Data R5b [646/1044]
8.338292	\longleftarrow		1452	Application Data R6 [1044], Application Data R7a [398/1044]
8.338386	\longrightarrow	ACK		
8.343465	\longleftarrow		646	Application Data R7b [646/1044]
8.377197	\longleftarrow		832	Application Data R8 [827]
8.377291	\longrightarrow	ACK		
8.379235	\longrightarrow		27	Alert - Close Notify [22]
8.379346	\longrightarrow	FIN, ACK		
8.456271	\longleftarrow	ACK		

4.6 *Client Certificate URL*

4.6.1 Especificação

```
enum {
    ..., certificate_url(21), ...,
    (255)
} HandshakeType;
```

Listagem 18: *Definição do novo tipo de mensagem de handshake Certificate URL*

After negotiation of the use of client certificate URLs has been successfully completed (by exchanging hellos including "client_certificate_url" extensions), clients MAY send a "CertificateURL" message in place of a "Certificate" message:

```
struct {
    CertChainType type;
    URLAndOptionalHash url_and_hash_list<1..2^16-1>;
} CertificateURL;

enum {
    individual_certs(0), pkipath(1), (255)
} CertChainType;

struct {
    opaque url<1..2^16-1>;
    Boolean hash_present;
    select (hash_present) {
        case false: struct {};
        case true: SHA1Hash;
    } hash;
} URLAndOptionalHash;

enum {
    false(0), true(1)
} Boolean;

opaque SHA1Hash[20];
```

Here "url_and_hash_list" contains a sequence of URLs and optional hashes.

When X.509 certificates are used, there are two possibilities:

- if CertificateURL.type is "individual_certs", each URL refers to a single DER-encoded X.509v3 certificate, with the URL for the client's certificate first, or
- if CertificateURL.type is "pkipath", the list contains a single URL referring to a DER-encoded certificate chain, using the type PkiPath described in Section 8.

Listagem 19: RFC 3546, trecho da seção 3.3

4.6.2 Divergências

1. Os certificados individuais podem estar tanto no formato PEM quanto DER (oficial). Foi implementada uma heurística simples para identificar o formato;
2. A aplicação servidora não está exigindo que o tipo MIME indicado na resposta HTTP seja application/pkix-pkipath, como dita a especificação.

4.6.3 API Estendida

```
/* ----- Client Side ----- */
void tlsx_cert_chain_empty (
    CERTIFICATE_CHAIN *cert_chain
);

int tlsx_cert_chain_add (
    CERTIFICATE_CHAIN *cert_chain,
    int cert_type,      /* TLSX_CERT_CHAIN_TYPE_INDIVIDUAL or
                        TLSX_CERT_CHAIN_TYPE_PKIPATH */
    const char *url,
    const unsigned char *shashash /* optional (may be NULL) */
);

void TLSX_set_certificate_chain (
    SSL *ssl,
    const CERTIFICATE_CHAIN *cert_chain
);

/* ----- Server Side ----- */
void TLSX_CTX_set_certificate_fetcher_callback (
    SSL_CTX *ctx,
    int (*cb) (
        SSL *ssl,
        void *uarg,
```

```
    const char *url,  
    unsigned char **data,  
    unsigned int *data_len  
),  
void *uarg  
);
```

Listagem 20: API Estendida para a extensão CCU

4.6.4 Implementação

Como a aplicação servidora, ao aceitar essa extensão, assume a responsabilidade de obter a cadeia de certificados do cliente, fez-se necessário embutir esse mecanismo na aplicação `s_server`. Optou-se pela criação de um novo módulo de nome “`wget.c`” que, através da biblioteca *open source libcurl* [27] implementa um cliente HTTP genérico. O pacote de desenvolvimento desta biblioteca (`libcurl3-dev`, versão 7.14.0, no ambiente Linux de testes) deve estar previamente instalado na máquina para *build* da aplicação `s_server`.

A geração dos certificados individuais em formato DER (ASN.1 binário) a partir do formato PEM (ASCII Base64) é trivialmente realizada através do comando:

```
openssl x509 -in cert.pem -out cert.der -outform DER
```

Listagem 21: Conversão de certificados para o formato DER

Como o formato denominado PKIPATH ainda não é nativamente suportado pelo OpenSSL e como também não se encontrou nenhuma ferramenta genérica que o suportasse, foi necessária a criação de uma pequena aplicação em Java 1.4 para realizar essa tarefa (ver Apêndice B na página 85).

A configuração do servidor HTTP (Apache) foi modificada para que o tipo `application/pkix-pkipath` seja retornado para os arquivos com extensão “.pki”.

`s3_clnt.c::ssl3_connect`

A máquina de estados foi modificada no tratamento do estado `SSL3_ST_CW_CERT_A` onde se decide se a nova mensagem *Certificate URL* será enviada no lugar da *Certificate* ou não.

`s3_srvr.c::ssl3_get_client_certificate`

Passou a aceitar e processar a mensagem *Certificate URL* caso tenha sido acordada no estágio inicial do *handshake*. Toda a decodificação dessa nova mensagem está embutida nesta função, delegando apenas a recuperação dos certificados para as funções `tlx_retrieve_certificate` e `tlx_retrieve_pkipath` conforme o caso.

4.6.5 Testes de Conformidade

Precisou-se, primeiramente, configurar a aplicação servidora para exibir a autenticação do cliente, passando a acioná-la como se segue:

```
./s_server.sh -Verify 2
```

Listagem 22: Chamada da *s_server.sh* para forçar a autenticação do cliente

A sintaxe usada na aplicação cliente para exercitar essa extensão foi:

```
./s_client.sh -tlsx_cert_url http://localhost/certs/client.crt \  
-key client.key
```

Listagem 23: Chamada da *s_client.sh* para teste da extensão CCU

Para capturar também o tráfego HTTP entre a aplicação *s_server* e o servidor Apache, a chamada do *tethereal* foi modificada para:

```
tethereal -i lo -V -x port 443 or port 80
```

Listagem 24: Chamada do *tethereal* para capturar tráfegos TLS e HTTP

4.6.5.1 Resultado do teste com certificados individuais

Novamente esbarramos em uma limitação do *tethereal* que não foi capaz de decodificar propriamente a nova mensagem *Certificate URL*, que a identificou erroneamente como *Encrypted Handshake Message*. Mas é possível comprovar a correção da implementação conferindo diretamente a sequência de bytes do registro correspondente a essa nova mensagem, que se encontra destacada na listagem a seguir:

Secure Socket Layer

```
TLS Record Layer: Handshake Protocol: Encrypted Handshake Message  
  Content Type: Handshake (22)  
  Version: TLS 1.0 (0x0301)  
  Length: 43  
  Handshake Protocol: Encrypted Handshake Message  
TLS Record Layer: Handshake Protocol: Client Key Exchange  
  Content Type: Handshake (22)  
  Version: TLS 1.0 (0x0301)  
  Length: 134  
  Handshake Protocol: Client Key Exchange  
    Handshake Type: Client Key Exchange (16)  
    Length: 130  
TLS Record Layer: Handshake Protocol: Certificate Verify  
  Content Type: Handshake (22)  
  Version: TLS 1.0 (0x0301)
```

```

Length: 134
Handshake Protocol: Certificate Verify
  Handshake Type: Certificate Verify (15)
  Length: 130
TLS Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
  Content Type: Change Cipher Spec (20)
  Version: TLS 1.0 (0x0301)
  Length: 1
  Change Cipher Spec Message
TLS Record Layer: Handshake Protocol: Encrypted Handshake Message
  Content Type: Handshake (22)
  Version: TLS 1.0 (0x0301)
  Length: 36
  Handshake Protocol: Encrypted Handshake Message

0000  00 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00  .....E.
0010  01 9d ac cf 40 00 40 06 8e 89 7f 00 00 01 7f 00  ....@.@.....
0020  00 01 c7 bb 01 bb fb 40 02 02 fb b0 f8 01 50 18  .....@.....P.
0030  7f ff d3 25 00 00 16 03 01 00 2b 15 00 00 27 00  ...%......+...'.
0040  00 24 00 21 68 74 74 70 3a 2f 2f 6c 6f 63 61 6c  .$.!http://local
0050  68 6f 73 74 2f 63 65 72 74 73 2f 63 6c 69 65 6e  8f.73742f63657274732f636c69656e
0060  74 2e 63 72 74 00 16 03 01 00 86 10 00 00 82 00  742e6372740016030100861000008200
0070  80 82 56 c1 53 4a 26 be f1 23 dd 3a 9d 76 19 8c  ..V.SJ&..#...v..
0080  ac 0e d0 e1 72 a3 0d 13 b0 59 cd 35 82 e0 ee 23  ....r....Y.5...#
0090  26 8e d7 61 71 09 1d 46 bd 01 86 a8 8b 57 0f e9  &..aq..F.....W..
                                .....
0180  01 01 16 03 01 00 24 7e c1 60 17 68 94 44 ae dd  ....~.'..h.D..
0190  1c 90 2c 40 52 28 42 54 4b 25 3c 9b d0 92 68 67  ..,@R(BTK%<...hg
01a0  d1 ca 9e 6c 13 62 0e d3 ed ae 51  ....l.b....

```

Listagem 25: Mensagem Certificate URL

```

Hypertext Transfer Protocol
GET /certs/client.crt HTTP/1.1\r\n
  Request Method: GET
  Request URI: /certs/client.crt
  Request Version: HTTP/1.1
Host: localhost\r\n
Accept: */*\r\n
\r\n

```

Listagem 26: Solicitação HTTP enviada pela aplicação s_server

```

Hypertext Transfer Protocol
HTTP/1.1 200 OK\r\n
  Request Version: HTTP/1.1

```



```
Response Code: 200
Date: Thu, 05 Jan 2006 17:32:37 GMT\r\n
Server: Apache/2.0.54 (Ubuntu) DAV/2 SVN/1.2.0 mod_python/3.1.3 Python
      /2.4.2 PHP/4.4.0-3 mod_ruby/1.2.4 Ruby/1.8.3(2005-06-23)\r\n
Last-Modified: Thu, 05 Jan 2006 17:01:44 GMT\r\n
ETag: "240f2-b66-7cbc2e00"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2918\r\n
Content-Type: application/x-x509-ca-cert\r\n
\r\n
```

Listagem 27: *Início (cabeçalho) da resposta HTTP enviada pelo servidor para a aplicação s_server*

4.6.5.2 Resultado do teste com PKIPATH

A sintaxe usada para aplicação cliente foi modificada para:

```
./s_client.sh -tlsx_pkipath_url http://localhost/certs/certs.pki \
  -key client.key
```

Valem as mesmas considerações feitas para o teste anterior quanto à análise dos dados capturados pelo *tethereal*.

Secure Socket Layer

```
TLS Record Layer: Handshake Protocol: Encrypted Handshake Message
  Content Type: Handshake (22)
  Version: TLS 1.0 (0x0301)
  Length: 42
  Handshake Protocol: Encrypted Handshake Message
TLS Record Layer: Handshake Protocol: Client Key Exchange
  Content Type: Handshake (22)
  Version: TLS 1.0 (0x0301)
  Length: 134
  Handshake Protocol: Client Key Exchange
    Handshake Type: Client Key Exchange (16)
    Length: 130
TLS Record Layer: Handshake Protocol: Certificate Verify
  Content Type: Handshake (22)
  Version: TLS 1.0 (0x0301)
  Length: 134
  Handshake Protocol: Certificate Verify
    Handshake Type: Certificate Verify (15)
    Length: 130
TLS Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
```

```

Content Type: Change Cipher Spec (20)
Version: TLS 1.0 (0x0301)
Length: 1
Change Cipher Spec Message
TLS Record Layer: Handshake Protocol: Encrypted Handshake Message
Content Type: Handshake (22)
Version: TLS 1.0 (0x0301)
Length: 36
Handshake Protocol: Encrypted Handshake Message

0000  00 00 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00  .....E.
0010  01 9c 6a 03 40 00 40 06 d1 56 7f 00 00 01 7f 00  ..j.@.@..V.....
0020  00 01 cd 70 01 bb 21 39 b1 cc 20 ef 7e 63 50 18  ...p..!9.. .~cP.
0030  7f ff d2 22 00 00 16 03 01 00 2a 15 00 00 26 01  ...".*****&.
0040  00 23 00 20 68 74 74 70 3a 2f 2f 6c 6f 63 61 6c  .#. http://local
0050  68 6f 73 74 2f 63 65 72 74 73 2f 63 65 72 74 73  host/certs/certs
0060  2e 70 6b 69 00 16 03 01 00 86 10 00 00 82 00 80  .pki.....
0070  7c fb e4 c7 8a 27 ef 23 2d 18 f8 be f2 97 d1 34  ....'..#-.....4
0080  30 b0 a3 4b 1f d8 b5 8e 49 d5 a6 eb 3e ea af 6d  0..K....I...>..m
0090  ae a1 0d 94 cb 66 89 b4 74 d2 86 5d 64 80 c0 96  ....f..t..]d...
      .....
0180  01 16 03 01 00 24 7e bc d4 51 8f a9 95 33 b6 0c  ....$~..Q...3..
0190  0b c2 db 9b 8d cc 56 28 81 14 a0 74 35 36 3e 8b  ....V(...t56>.
01a0  22 9c f1 f1 0d d3 3e 0f fe 3b  ".....>..

```

Listagem 28: *Mensagem Certificate URL*

```

Hypertext Transfer Protocol
GET /certs/certs.pki HTTP/1.1\r\n
Request Method: GET
Request URI: /certs/certs.pki
Request Version: HTTP/1.1
Host: localhost\r\n
Accept: */*\r\n
\r\n

```

Listagem 29: *Solicitação HTTP enviada pela aplicação s_server*

```

Hypertext Transfer Protocol
HTTP/1.1 200 OK\r\n
Request Version: HTTP/1.1
Response Code: 200
Date: Thu, 05 Jan 2006 17:49:30 GMT\r\n
Server: Apache/2.0.54 (Ubuntu) DAV/2 SVN/1.2.0 mod_python/3.1.3 Python
/2.4.2 PHP/4.4.0-3 mod_ruby/1.2.4 Ruby/1.8.3(2005-06-23)\r\n
Last-Modified: Thu, 05 Jan 2006 16:42:45 GMT\r\n

```

```
ETag: "240f0-654-38d86b40"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 1620\r\n
Content-Type: application/pkix-pkipath\r\n
\r\n
```

Listagem 30: *Início (cabeçalho) da resposta HTTP enviada pelo servidor para a aplicação `s_server`*

4.7 *Truncated HMAC*

4.7.1 Especificação

The "extension_data" field of this extension SHALL be empty.

Listagem 31: RFC 3546, trecho da seção 3.5

4.7.2 API Estendida

```
void TLSX_enable_truncated_hmac (  
    SSL *ssl  
);
```

Listagem 32: API Estendida para a extensão TMAC

4.7.3 Divergências

Nenhuma.

4.7.4 Implementação

s3_clnt.c::ssl3_connect

A máquina de estados foi modificada no tratamento do estado SSL_ST_OK quando o parâmetro `tlsx_truncated_hmac` é efetivado se assim foi acordado com o servidor.

s3_pkt.c::ssl3_get_record

Modificada para reduzir o tamanho do *Hash-based Message Authentication Code* da mensagem a ser validada para `TLSX_TRUNCATED_HMAC_SIZE` (10) se assim foi acordado.

s3_pkt.c::do_ssl3_write

Modificada para reduzir o tamanho do *Hash-based Message Authentication Code* da mensagem a ser escrita para `TLSX_TRUNCATED_HMAC_SIZE` (10) se assim foi acordado.

s3_srvr.c::ssl3_accept

No estado SSL_ST_OK quando o parâmetro `tlsx_truncated_hmac` é efetivado se assim foi acordado com o cliente;

4.7.5 Testes de Conformidade

Além de demonstrar a redução no tamanho do HMAC, os testes foram projetados para comprovar que essa redução só é efetiva após a conclusão do *handshake* (não afetando a mensagem *Finished*, portanto) e que esse parâmetro volta a ser efetivo após a restauração da sessão.

A sintaxe usada na chamada do cliente `s_client` foi:

```
./s_client.sh -tlsex_truncated_hmac
```

Duas conexões TLS foram estabelecidas com o seguinte diálogo entre as aplicações `s_client` e `s_server`:

Tabela 9: *Diálogo entre as aplicações `s_client` e `s_server` para testar a eficácia da extensão TMAC*

s_client	s_server
request 1	
	response 1
N	
request 2	
	response 2
Q	

Todo o tráfego dessas conexões foi capturado e encontra-se sintetizado nas Tabelas 10 e 11:

```
Secure Socket Layer
SSL Record Layer: Handshake Protocol: Client Hello
  Content Type: Handshake (22)
  Version: TLS 1.0 (0x0301)
  Length: 51
  Handshake Protocol: Client Hello
    Handshake Type: Client Hello (1)
    Length: 47
    Version: TLS 1.0 (0x0301)
    Random.gmt_unix_time: Jan  6, 2006 09:10:53.000000000
    Random.bytes
    Session ID Length: 0
    Cipher Suites Length: 2
    Cipher Suites (1 suite)
    Cipher Suite: TLS_RSA_WITH_RC4_128_SHA (0x0005)
    Compression Methods Length: 1
    Compression Methods (1 method)
```

Tabela 10: Primeira conexão (nova): Registros com HMAC reduzido

Time (s)	Direction C \longleftrightarrow S	TCP		TLS Record [length]
		Flags	Length	
0.000000	\longrightarrow	SYN		
0.145307	\longleftarrow	SYN, ACK		
0.145435	\longrightarrow	ACK		
0.148974	\longrightarrow		56	Client Hello [51]
1.897516	\longleftarrow		1452	Server Hello [80], Certificate [1362/1627]
0.414415	\longrightarrow	ACK		
0.415191	\longleftarrow		274	Certificate [265/1627], Server Hello Done [4]
0.415762	\longrightarrow	ACK		
0.428106	\longrightarrow		186	Client Key Exchange [134], Change Cipher Spec [1], Finished [36]
0.627651	\longleftarrow		47	Change Cipher Spec [1], Finished [36]
0.667665	\longrightarrow	ACK		
3.680601	\longrightarrow		25	Application Data [20]
3.856378	\longleftarrow	ACK		
9.955240	\longleftarrow		26	Application Data [21]
9.955423	\longrightarrow	ACK		
13.803550	\longrightarrow		17	Alert - Close Notify [12]
13.803895	\longrightarrow	FIN, ACK		
13.842239	\longleftarrow	ACK		
13.851260	\longleftarrow	FIN, ACK		
13.851370	\longrightarrow	ACK		

```

Compression Method: null (0)
Extensions Length: 4
Extension: truncated_hmac
Type: truncated_hmac (0x0004)
Length: 0
Data (0 bytes)

```

Listagem 33: Detalhe do Client Hello estendido

Tabela 11: *Primeira conexão (restaurada): Registros com HMAC reduzido*

Time (s)	Direction C \longleftrightarrow S	TCP		TLS Record [length]
		Flags	Length	
13.804240	\longrightarrow	SYN		
13.845121	\longleftarrow	SYN, ACK		
13.845296	\longrightarrow	ACK		
13.846581	\longrightarrow		88	Client Hello [83]
13.956312	\longleftarrow		126	Server Hello [74], Change Cipher Spec [1], Finished [36]
13.957341	\longrightarrow	ACK		
13.959072	\longrightarrow		47	Change Cipher Spec [1], Finished [36]
14.088635	\longleftarrow	ACK		
17.725468	\longrightarrow		25	Application Data [20]
17.902308	\longleftarrow	ACK		
21.159691	\longleftarrow		26	Application Data [21]
21.199546	\longrightarrow	ACK		
22.834441	\longrightarrow		17	Alert - Close Notify [12]
22.834557	\longrightarrow	FIN, ACK		

4.8 *Trusted CA Indication*

4.8.1 Especificação

The "extension_data" field of this extension SHALL contain "TrustedAuthorities" where:

```
struct {
    TrustedAuthority trusted_authorities_list<0..2^16-1>;
} TrustedAuthorities;

struct {
    IdentifierType identifier_type;
    select (identifier_type) {
        case pre_agreed: struct {};
        case key_sha1_hash: SHA1Hash;
        case x509_name: DistinguishedName;
        case cert_sha1_hash: SHA1Hash;
    } identifier;
} TrustedAuthority;

enum {
    pre_agreed(0), key_sha1_hash(1), x509_name(2),
    cert_sha1_hash(3), (255)
} IdentifierType;

opaque DistinguishedName<1..2^16-1>;
```

Here "TrustedAuthorities" provides a list of CA root key identifiers that the client possesses. Each CA root key is identified via either:

- "pre_agreed" - no CA root key identity supplied.
- "key_sha1_hash" - contains the SHA-1 hash of the CA root key. For DSA and ECDSA keys, this is the hash of the "subjectPublicKey" value. For RSA keys, the hash is of the big-endian byte string representation of the modulus without any initial 0-valued bytes. (This copies the key hash formats deployed in other environments.)
- "x509_name" - contains the DER-encoded X.509 DistinguishedName of the CA.
- "cert_sha1_hash" - contains the SHA-1 hash of a DER-encoded

Certificate containing the CA root key.

Listagem 34: RFC 3546, trecho da seção 3.4

4.8.2 API Estendida

```
void TLSX_set_trusted_ca (
    SSL *ssl,
    int id_type,      /* TLSX_CERT_ID_PRE_AGREED,
                      TLSX_CERT_ID_KEY_SHA1_HASH,
                      TLSX_CERT_ID_X509_NAME, or
                      TLSX_CERT_ID_CERT_SHA1_HASH */
    const char *CAfile, /* optional */
    const char *CApath  /* optional */
);
```

Listagem 35: API Estendida para a extensão TCI

Essa função associa ao objeto SSL uma lista de IDs de todos os certificados encontrados no diretório indicado por CApath cujo nome base (sem diretório) adere ao padrão utilizado pelo OpenSSL para certificados de CAs, ou seja, segue o formato “hhhhhhh.d”, onde “hhhhhhh” é uma sequência de oito dígitos hexadecimais e “d” é um dígito decimal. O certificado opcionalmente indicado por CAfile é também acrescentado a esse conjunto. Todos esses certificados devem estar em formato PEM (Base-64).

4.8.3 Divergências

1. Não foi possível implementar a parte servidora dessa extensão pois o OpenSSL não permite a configuração de um servidor com múltiplos certificados diferenciados apenas pelo emissor;
2. Como normalmente as aplicações que utilizam o OpenSSL especificam o *Certificate Store* coletivamente (via SSL_CTX_load_verify_locations), a API estendida obriga a adoção de um mesmo tipo de identificador para todos os certificados.

4.8.4 Testes de Conformidade

Como os resultados são bastante semelhantes para os diversos tipos de IDs, apresenta-se aqui somente o resultado de teste efetuado com o tipo TLSX_CERT_ID_CERT_SHA1_HASH com seguinte sintaxe na execução da aplicação s_client:

```
./s_client.sh -tlsx_trusted_ca_id cert_sha1_hash
```

O diretório de trabalho possuía somente um certificado cujo *hash* SHA-1 pôde ser obtido diretamente com o comando shasum:

```
ebo@esparta:~/openssl-tlsx$ shasum 3ff23a0a.d
27399b0a3e46c936852cc3662b90d0e4d8635e78 3ff23a0a.d
```

Listagem 36: Cálculo do hash SHA-1 do certificado do CA

A equivalência do *hash* obtido com esse comando e o enviado na mensagem *Client Hello* abaixo (destacado em negrito e itálico no final da listagem), comprova a correção da implementação desta extensão.

Secure Socket Layer

```
SSL Record Layer: Handshake Protocol: Client Hello
Content Type: Handshake (22)
Version: TLS 1.0 (0x0301)
Length: 72
Handshake Protocol: Client Hello
Handshake Type: Client Hello (1)
Length: 68
Version: TLS 1.0 (0x0301)
Random.gmt_unix_time: Jan  5, 2006 16:20:24.000000000
Random.bytes
Session ID Length: 0
Cipher Suites Length: 2
Cipher Suites (1 suite)
Cipher Suite: TLS_RSA_WITH_RC4_128_SHA (0x0005)
Compression Methods Length: 1
Compression Methods (1 method)
Compression Method: null (0)
Extensions Length: 25
Extension: trusted_ca_keys
Type: trusted_ca_keys (0x0003)
Length: 21
Data (21 bytes)
```

```
0000  00 00 00 00 00 00 00 00 00 00 00 00 00 08 00 45 00  .....E.
0010  00 75 22 ee 40 00 40 06 19 93 7f 00 00 01 7f 00  .....u".@.@.....
0020  00 01 c7 a9 01 bb eb ae f5 b1 eb 59 e3 a9 50 18  .....Y..P.
0030  7f ff 72 72 00 00 16 03 01 00 48 01 00 00 44 03  ..rr.....H...D.
0040  01 43 bd 63 68 29 5c 97 3f 14 37 d7 76 c0 11 35  .C.ch)\.?.7.v..5
0050  6e 25 fc bd 6a d8 91 e9 1d b9 2e 85 cf 39 38 56  n%..j.....98V
0060  bb 00 00 02 00 05 01 00 00 19 00 03 00 15 03 27  .....'.
0070  39 9b 0a 3e 46 c9 36 85 2c c3 66 2b 90 d0 e4 d8  9..>F.6.,.f+....
0080  63 5e 78                                     c^x
```

Listagem 37: Mensagem *Client Hello* contendo a extensão TCI

4.9 Server Certificate Chain Indication

4.9.1 Especificação

```
enum {  
    ..., certificate_chain(10), (65535)  
} ExtensionType;
```

Listagem 38: Definição (não-oficial) da extensão SCCI

The "extension_data" field of this extension SHALL contain "CertificateIdentifiers" where:

```
struct {  
    CertificateIdentifier certificate_identifiers_list<0..2^16-1>;  
} CertificateIdentifiers;
```

CertificateIdentifier has the same definition of TrustedAuthority.

Listagem 39: Especificação (não-oficial) da extensão SCCI

4.9.2 API Estendida

```
/* Client Side */  
STACK_OF(X509) *TLSX_get_cert_chain (  
    SSL *ssl  
);  
  
void TLSX_set_server_cert_chain (  
    SSL *ssl,  
    STACK_OF(X509) *cert_chain,  
    int id_type /* TLSX_CERT_ID_PRE_AGREED,  
                TLSX_CERT_ID_KEY_SHA1_HASH,  
                TLSX_CERT_ID_X509_NAME, or  
                TLSX_CERT_ID_CERT_SHA1_HASH */  
);  
  
void TLSX_free_cert_chain (  
    STACK_OF(X509) **chain  
);
```

Listagem 40: API Estendida para a extensão SCCI

A `TLSX_get_cert_chain` deve ser usada em qualquer momento após o estabelecimento da conexão TLS, mas antes do fechamento da mesma. Ela devolve uma cópia

de toda a cadeia de certificados do servidor que poderá então ser usada na próxima conexão a este mesmo servidor. A função `TLSX_set_server_cert_chain` atribui uma referência a esta cadeia ao objeto SSL e ativa automaticamente a extensão *Server Certificate Chain Indication*.

A função `TLSX_free_cert_chain` libera toda a memória utilizada por essa cadeia quando não estiver mais sendo referenciada por nenhum objeto SSL. Essa referência será anulada quando da destruição do objeto SSL associado ou explicitamente através da chamada `TLSX_set_server_cert_chain(ssl, NULL, 0)`.

4.9.3 Divergências

Nenhuma por não haver uma norma oficial.

4.9.4 Implementação

`s3_both.c::ssl3_output_cert_chain`

Na versão original criava e serializava toda uma cadeia de certificados diretamente no *buffer* de saída, a partir do certificado final. Foi modificada para construir a cadeia e ao final decidir se esta será serializada ou será inteiramente suprimida, o que ocorrerá somente se as seguintes condições forem atendidas:

1. O papel da conexão for “servidor”;
2. A extensão *Server Certificate Chain Indication* tiver sido proposta e aceita;
3. Houver uma correspondência integral entre os IDs informados na extensão e os certificados selecionados.

`s3_clnt.c::ssl3_get_server_certificate`

Caso a mensagem *Certificate* oriunda do servidor esteja vazia e a extensão *Server Certificate Chain Indication* tiver sido acordada, a cadeia de certificados será simplesmente uma cópia da cadeia definida através da função `TLSX_set_server_cert_chain`. Nesse ponto é também incluída essa cadeia no cômputo do *hash* da mensagem *Finished* enviada ao final do *handshake*.

`s3_srvr.c::ssl3_accept`

Modificada para atualizar o *hash* a ser enviado na mensagem *Finished* com toda a cadeia de certificados caso esta tenha sido omitida na `ssl3_output_cert_chain`.

4.9.5 Testes de Conformidade

O teste foi realizado com a seguinte execução da aplicação `s_client`:

```
./s_client.sh -tlsx_server_cert_id cert_sha1_hash
```

Listagem 41: Chamada da *s_client.sh* para o teste da extensão SCCI

Uma vez acionada, entrou-se com a string “n” para forçar uma segunda conexão TLS sem *session resumption*. As mensagens TLS dessa última conexão e que foram impactadas por essa extensão encontram-se exibidas a seguir.

Secure Socket Layer

SSL Record Layer: Handshake Protocol: Client Hello

Content Type: Handshake (22)

Version: TLS 1.0 (0x0301)

Length: 93

Handshake Protocol: Client Hello

Handshake Type: Client Hello (1)

Length: 89

Version: TLS 1.0 (0x0301)

Random.gmt_unix_time: Jan 6, 2006 15:54:17.000000000

Random.bytes

Session ID Length: 0

Cipher Suites Length: 2

Cipher Suites (1 suite)

Cipher Suite: TLS_RSA_WITH_RC4_128_SHA (0x0005)

Compression Methods Length: 1

Compression Methods (1 method)

Compression Method: null (0)

Extensions Length: 46

Extension: Unknown 10

Type: Unknown (0x000a)

Length: 42

Data (42 bytes)

```
0000  00 c0 49 43 0b 55 00 50 04 aa 26 05 08 00 45 00  ..IC.U.P..&...E.
0010  00 8a 93 d1 40 00 40 06 c7 5a 0a 04 02 14 c9 2d  ....@.@..Z.....-
0020  09 fd a1 49 01 bb c6 bc e9 53 a6 d2 a6 84 50 18  ...I.....S....P.
0030  16 d0 3c 21 00 00 16 03 01 00 5d 01 00 00 59 03  ..<!......]...Y.
0040  01 43 be ae c9 59 97 28 3d 2c ef 3e d6 73 92 cd  .C...Y.(=,>.s...
0050  6a 83 e0 2f 0b ef b1 d1 46 8d 07 b1 6a f5 58 97  j../....F...j.X.
0060  a9 00 00 02 00 05 01 00 00 2e 00 0a 00 2a 03 0d  .....*...
0070  5c 3c bb 76 60 b4 36 9b a3 11 16 db 87 bd c0 81  \<.v'.6.....
0080  c1 14 da 03 27 39 9b 0a 3e 46 c9 36 85 2c c3 66  ....'9...>F.6.,.f
0090  2b 90 d0 e4 d8 63 5e 78  +....c^x
```

Listagem 42: Mensagem Client Hello com a extensão SCCI

Secure Socket Layer

```
TLS Record Layer: Handshake Protocol: Server Hello
  Content Type: Handshake (22)
  Version: TLS 1.0 (0x0301)
  Length: 80
  Handshake Protocol: Server Hello
    Handshake Type: Server Hello (2)
    Length: 76
    Version: TLS 1.0 (0x0301)
    Random.gmt_unix_time: Jan  6, 2006 15:54:09.000000000
    Random.bytes
    Session ID Length: 32
    Session ID (32 bytes)
    Cipher Suite: TLS_RSA_WITH_RC4_128_SHA (0x0005)
    Compression Method: null (0)
    Extensions Length: 4
    Extension: Unknown 10
  Type: Unknown (0x000a)
  Length: 0
  Data (0 bytes)
TLS Record Layer: Handshake Protocol: Certificate
  Content Type: Handshake (22)
  Version: TLS 1.0 (0x0301)
  Length: 7
  Handshake Protocol: Certificate
    Handshake Type: Certificate (11)
    Length: 3
    Certificates Length: 0
TLS Record Layer: Handshake Protocol: Server Hello Done
  Content Type: Handshake (22)
  Version: TLS 1.0 (0x0301)
  Length: 4
  Handshake Protocol: Server Hello Done
    Handshake Type: Server Hello Done (14)
    Length: 0
```

Listagem 43: Resposta do servidor com destaque para a mensagem Certificate vazia

Conclusão e Trabalhos Futuros

5.1 Conclusão

Apesar da RFC 3546 ter sido publicada há quase três anos, somente agora tem surgido implementações das extensões propostas, mas mesmo assim parciais.

A versão em desenvolvimento do OpenSSL (0.9.9) já contém uma implementação funcional da extensão *Server Name Indication* que também é contemplada na vindoura versão do *browser* Opera (9.0) e na pilha GnuTLS.

As demais extensões, todas voltadas para sistemas embutidos, tem recebido pouca ou quase nenhuma atenção dos desenvolvedores de *software open-source*, assim como de empresas comerciais.

Com base na minha experiência em desenvolvimento de soluções de comunicação, entendo que esse interesse tem sido atenuado por alguns mitos:

- De que o ostensivo e contínuo aumento na banda em praticamente todos os meios de comunicação dispensaria a necessidade de esforços para a otimização de protocolos. Essa expectativa não procede pois o aumento na demanda tem sido ainda mais vertiginoso, principalmente em redes utilizadas por equipamentos móveis;
- Que o mecanismo de restauração de sessões TLS anularia a aplicabilidade da maioria destas extensões. De fato, com excessão das extensões *Maximum Fragment Length* e *Truncated HMAC*, todas as outras cinco só são aplicáveis em *handshakes* completos.

No entanto, ao contrário do muitos supõem, em geral as sessões TLS não são mantidas por muito tempo pelos servidores HTTPS – os maiores beneficiados por este mecanismo – por causa do espaço em memória que seria necessário para armazená-las em larga escala. No site *Amazon.com*, por exemplo, as sessões TLS são descartadas após breves 2 minutos!

A minha real expectativa é de que a implementação alvo deste projeto possa contribuir efetivamente para popularizar o uso dessas extensões para um uso mais racional dos recursos disponíveis para sistemas restritos.

5.2 Propostas de Trabalhos Futuros

5.2.1 Tarefas Gerais

- Submissão ao *core team* de desenvolvedores do OpenSSL a fim de oficializar esta implementação;
- Replicar a API proposta no nível CTX, visando a consistência com a API oficial do OpenSSL;
- Garantir que as novas implementações são *thread-safe*;
- Incorporação de outras possíveis extensões já definidas pelo *IETF TLS Working Group* ou que ainda estejam em fase de discussão. Exemplo: a que permite a substituição do SHA-1 por algum algoritmo de *hashing* mais seguro (SHA-256 ou SHA-512, por exemplo);
- Implementar estas extensões em uma pilha SSL/TLS voltada para sistemas embutidos. O candidato mais adequado parece ser o MatrixSSL (<http://www.matrixssl.org/>), principalmente por causa do seu duplo-licenciamento (GPL e comercial);
- Corrigir o Ethereum para decodificar corretamente a mensagem *Certificate URL* e ser capaz de desfragmentar registros;

5.2.2 Extensão *Server Name Indication*

- Implementar a parte servidora, incluindo as necessárias modificações no “*mod_ssl*” (<http://httpd.apache.org/docs/2.0/ssl/>) para ativar e configurar os seus diversos parâmetros.

5.2.3 Extensão *Client Certificate URL*

- Implementar o *cache* de certificados no servidor;

- Analisar a possibilidade de usar algum esquema específico em que o cliente informe somente o contexto (ou *domain/realm*) e um identificador (*serial number/username*) e que o servidor seja capaz de “montar” uma URL com base nessas informações e em *template* configurado;
- Implementar mecanismos que restrinjam o acesso a um conjunto limitado de provedores de certificados. Visar também minimizar eventuais ataques do tipo *Denial of Service* (DoS).

5.2.4 Extensão *Certificate Status Request*

- Além de implementar esta extensão, avaliar a possibilidade do uso de outros protocolos alternativos, tais como *Simple Certificate Validation Protocol* (SCVP), *Data Certification Server* (DCS) e *Online Certificate Status Protocol Extensions* (OCSP-X);

5.2.5 Extensão *Server Certificate Chain Indication*

- Obtenção de um identificador oficial para esta extensão junto a IANA/IETF (vide <http://www.iana.org/assignments/tls-extensiontype-values>);
- Submissão desta nova extensão ao grupo de trabalho do IETF responsável pelo TLS para uma possível criação de uma *Internet-Draft* e futuramente de uma RFC;



Header “tlsx.h”

```
1 #ifndef HEADER_TLSX_H
2 #define HEADER_TLSX_H
3
4 #ifdef __cplusplus
5 extern "C" {
6 #endif
7
8 #define TLSX_SERVER_NAME_INDICATION      0
9 #define TLSX_MAX_FRAGMENT_LENGTH        1
10 #define TLSX_CLIENT_CERTIFICATE_URL      2
11 #define TLSX_TRUSTED_CA_IDS              3
12 #define TLSX_TRUNCATED_HMAC              4
13 #define TLSX_CERTIFICATE_STATUS_REQUEST  5
14 #define TLSX_SERVER_CERTIFICATE_ID       10
15
16 #define TLSX_LAST_EXTENSION               TLSX_SERVER_CERTIFICATE_ID
17
18 #define TLSX_MASK(e)                     (1 << (e))
19 #define TLSX_ALL_EXTENSIONS               (TLSX_MASK (TLSX_LAST_EXTENSION + 1) -
20     1)
21
22 #define MAX_HOSTNAME_LEN                 255
23
24 #define MAX_URL_LEN                      255
25
26 #define TLSX_CERT_CHAIN_TYPE_INDIVIDUAL  0
27 #define TLSX_CERT_CHAIN_TYPE_PKIPATH     1
28
29 #define TLSX_CERT_ID_PRE_AGREED          0
30 #define TLSX_CERT_ID_KEY_SHA1_HASH       1
31 #define TLSX_CERT_ID_X509_NAME           2
32 #define TLSX_CERT_ID_CERT_SHA1_HASH      3
33
34 #define TLSX_TRUNCATED_HMAC_SIZE          10
35
36 /*
37  * Extended Alert Descriptions.
```

```

37  */
38  #define TLSX_AD_UNSUPPORTED_EXTENSION          110
39  #define TLSX_AD_CERTIFICATE_UNOBTAINABLE      111
40  #define TLSX_AD_UNRECOGNIZED_NAME            112
41  #define TLSX_AD_BAD_CERTIFICATE_STATUS_RESPONSE 113
42  #define TLSX_AD_BAD_CERTIFICATE_HASH_VALUE    114
43
44  typedef struct certificate_url_st
45  {
46      char url [MAX_URL_LEN+1];
47      int hash_present;
48      unsigned char shalhash [SHA_DIGEST_LENGTH];
49  } CERTIFICATE_URL;
50
51  #define SK_CERTIFICATE_URL          STACK_OF(CERTIFICATE_URL)
52
53  typedef struct
54  {
55      int chain_type;
56      SK_CERTIFICATE_URL *chain;
57  } CERTIFICATE_CHAIN;
58
59  typedef struct
60  {
61      int type;
62      union
63      {
64          BUF_MEM *subject;
65          unsigned char shalhash [SHA_DIGEST_LENGTH];
66      } value;
67  } CERT_ID;
68
69  #define SK_CERT_ID          STACK_OF(CERT_ID)
70  #define SK_X509             STACK_OF(X509)
71
72  typedef struct
73  {
74      /* Common variables. */
75      int agreeded;          /* proposed and accepted extensions. */
76
77      /* Client-side variables. */
78      int wanted;            /* Intended extensions. */
79      int required;          /* Indicates which extensions are mandatory. */
80      char *servername;
81      unsigned int max_fragment_length_id;
82      CERTIFICATE_CHAIN certificate_chain;
83      SK_CERT_ID *trusted_ca_list;
84      int truncated_hmac;
85      SK_X509 *server_cert_chain;
86      int server_cert_id_type;

```

```

87
88     /* Server-side variables. */
89     int allowed;           /* Permitted extensions. */
90     int (*servername_cb) (SSL *, const char *, void *);
91     void *servername_arg;
92     int (*certificate_fetcher_cb) (SSL *, void *, const char *, unsigned char **,
93         unsigned int *);
94     void *certificate_fetcher_arg;
95     SK_CERT_ID *cert_id_list;    /* IDs received with the ClientHello message. */
96     } TLS_EXTENSIONS;
97 /*
98  * Client-side API.
99  */
100 void TLSX_CTX_set_maximum_fragment_length(SSL_CTX *ctx, int frag_length);
101 void TLSX_CTX_set_servername_callback(SSL_CTX *ctx, int (*cb) (SSL *, const char *,
102     void *), void *arg);
103 void TLSX_CTX_set_certificate_fetcher_callback(SSL_CTX *ctx, int (*cb) (SSL *, void
104     *, const char *, unsigned char **, unsigned int *), void *arg);
105 void TLSX_CTX_allow (SSL_CTX *ctx, int mask);
106
107 void TLSX_set_servername(SSL *ssl, const char *server_name);
108
109 void TLSX_set_maximum_fragment_length(SSL *ssl, int frag_length);
110
111 void TLSX_cert_chain_empty(CERTIFICATE_CHAIN *cert_chain);
112 int TLSX_cert_chain_add(CERTIFICATE_CHAIN *cert_chain, int cert_type, const char *
113     url, const unsigned char *shashash);
114 void TLSX_set_certificate_chain(SSL *ssl, const CERTIFICATE_CHAIN *cert_chain);
115
116 void TLSX_set_trusted_ca(SSL *ssl, int id_type, const char *CAfile, const char *
117     CApth);
118
119 void TLSX_enable_truncated_hmac(SSL *ssl);
120
121 SK_X509 *TLSX_get_cert_chain(SSL *ssl);
122 void TLSX_free_cert_chain(SK_X509 **chain);
123 void TLSX_set_server_cert_chain(SSL *ssl, SK_X509 *cert_chain, int id_type);
124
125 /*
126  * Server-side API.
127  */
128 void TLSX_allow (SSL *ssl, int mask);
129
130 #ifdef __cplusplus
131 }
132 #endif
133 #endif

```

Listagem 44: Header *tlsx.h*

B

Programa “MakePkiPath.java”

```
import java.util.Iterator;
import java.util.List;
import java.util.ArrayList;
import java.security.cert.X509Certificate;
import java.security.cert.CertificateFactory;
import java.security.cert.CertPath;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.BufferedInputStream;

public class MakePkiPath
{
    private static void display (String str)
    {
        System.out.println (str);
    }

    public static void main (String [] args) throws Exception
    {
        if (args.length < 1)
        {
            display ("Usage:: java MakePkiPath <certfile>");
            System.exit (1);
        }

        CertificateFactory cf = CertificateFactory.getInstance ("X.509");
        List certs = new ArrayList ();

        for (int i = 0; i < args.length; i++)
        {
            FileInputStream fis = new FileInputStream (args [i]);
```

```
        BufferedInputStream bis = new BufferedInputStream (fis);

        while (bis.available () > 0)
        {
            certs.add (cf.generateCertificate (bis));
        }
    }

    CertPath cp = cf.generateCertPath (certs);

    // print each certificate in the path
    Iterator i = cp.getCertificates ().iterator ();
    while (i.hasNext ())
    {
        X509Certificate cert = (X509Certificate) i.next ();
        display (cert.toString ());
    }

    display ("Creatint PkiPath");
    FileOutputStream fos = new FileOutputStream ("certs.pki");
    fos.write (cp.getEncoded ("PkiPath"));
    fos.close ();
}
}
```

Listagem 45: Programa *MakePkiPath.java*

Referências Bibliográficas

- 1 DIERKS, T.; ALLEN, C. *RFC 2246: The TLS Protocol Version 1.0*. RFC Editor, 1999.
<http://www.ietf.org/rfc/rfc2246.txt>.
- 2 BLAKE-WILSON, S. et al. *RFC 3546: Transport Layer Security (TLS) Extensions*. 2003.
<http://www.ietf.org/rfc/rfc3546.txt>.
- 3 STEFFEN, A. *Sichere Netzwerkkommunikation - Overview*. 2004.
<http://www-t.zhwin.ch/it/snk/>.
- 4 MENEZES, A. J.; OORSCHOT, P. C. V.; VANSTONE, S. A. *Handbook of Applied Cryptography*. Boca Raton, FL, USA: CRC Press, 1996. ISBN 0-8493-8523-7.
<http://www.cacr.math.uwaterloo.ca/hac/>.
- 5 RIVEST, R. *RFC 1321: The MD5 Message-Digest Algorithm*. 1992.
<http://www.ietf.org/rfc/rfc1321.txt>.
- 6 NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. *FIPS PUB 180-1: Secure Hash Standard*. 1995.
<http://www.itl.nist.gov/fipspubs/fip180-1.htm>.
- 7 KRAWCZYK, H.; BELLARE, M.; CANETTI, R. *RFC 2104: HMAC: Keyed-Hashing for Message Authentication*. 1997.
<http://www.ietf.org/rfc/rfc2104.txt>.
- 8 KAUKONEN, K.; THAYER, R. *A Stream Cipher Encryption Algorithm*. 1999.
<http://www.mozilla.org/projects/security/pki/nss-/draft-kaukonen-cipher-arcfour-03.txt>.
- 9 NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. *FIPS PUB 46-2: Data Encryption Standard*. 1988.
<http://www.itl.nist.gov/fipspubs/fip46-2.htm>.
- 10 AMERICAN NATIONAL STANDARDS INSTITUTE. *ANSI draft X9.52, Triple Data Encryption Algorithm Modes of Operation*. 1996.
- 11 RIVEST, R. *RFC 2268: A Description of the RC2 Encryption Algorithm*. 1998.
<http://www.ietf.org/rfc/rfc2268.txt>.
- 12 LAI, X. *On the Design and Security of Block Ciphers*. Hartung-Gorre, Germany: Konstanz, 1992. (ETH Series in Information Processing). ISBN 3-89191-573-X.

- 13 ADAMS, C.; LLOYD, S. *Understanding PKI: Concepts, Standards, and Deployment Considerations*. 2nd.. ed. Boston, MA, USA: Addison Wesley, 2002. ISBN 0-672-32391-5.
- 14 RIVEST, R. L.; SHAMIR, A.; ADLEMAN, L. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, ACM Press, New York, NY, USA, v. 21, n. 2, p. 120–126, 1978. ISSN 0001-0782.
- 15 NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. *FIPS PUB 186: Digital Signature Standard*. 1994.
<http://www.itl.nist.gov/fipspubs/fip186.htm>.
- 16 DIFFIE, W.; HELLMAN, M. E. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22, n. 6, p. 644–654, 1976.
<http://www-ee.stanford.edu/~hellman/publications/24.pdf>.
- 17 HOUSLEY, R. et al. *RFC 3280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC Editor, 2002.
<http://www.ietf.org/rfc/rfc3280.txt>.
- 18 RESCORLA, E. *SSL and TLS: Designing and Building Secure Systems*. Boston, MA, USA: Addison-Wesley, 2000. ISBN 0-201-61598-3.
- 19 THOMAS, S. *SSL and TLS Essentials - Securing the Web*. New York, NY, USA: John Wiley & Sons, 2000. ISBN 0-471-38354-6.
- 20 FIELDING, R. T. et al. *RFC 2616: Hypertext Transfer Protocol – HTTP/1.1*. RFC Editor, 1999.
<http://www.ietf.org/rfc/rfc2616.txt>.
- 21 MYERS, M. et al. *RFC 2560: X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP*. RFC Editor, 1999.
<http://www.ietf.org/rfc/rfc2560.txt>.
- 22 THE OPENSSL CORE TEAM. *OpenSSL*. 2005.
<http://www.openssl.org/>.
- 23 VIEGA, J.; MESSIER, M.; CHANDRA, P. *Network Security with OpenSSL*. Sebastopol, CA, USA: O'Reilly & Associates, 2002. ISBN 0-596-00270-X.
<http://www.opensslbook.com/>.
- 24 COMBS, G. et al. *Ethereal: A Network Protocol Analyzer*. 2005.
<http://www.ethereal.com>.
- 25 FREE SOFTWARE FOUNDATION. *GnuTLS: The GNU Transport Layer Security Library*. 2005.
<http://www.gnu.org/software/gnutls/>.

- 26 OPERA SOFTWARE ASA. *Opera 9 Technology Preview 2*. 2006.
<http://labs.opera.com/downloads/>.
- 27 STENBERG, D. *libcurl: the multiprotocol file transfer library*. 2005.
<http://curl.haxx.se/libcurl/>.