Love all, trust few: On trusting intermediaries in HTTP

Thomas Fossati
Alcatel-Lucent
Cambridge (UK)
thomas.fossati@alcatel-lucent.com

Vijay K. Gurbani
Alcatel-Lucent, Bell Labs
Naperville, IL 60566 (USA)
vkg@bell-labs.com

Vladimir Kolesnikov
Alcatel-Lucent, Bell Labs
Murray Hill, NJ 07974 (USA)
kolesnikov@research.bell-labs.com

ABSTRACT

Recent pervasive monitoring of Internet traffic has resulted in an effort to protect all communications by using Transport Layer Security (TLS) to thwart malicious third parties. We argue that such large-scale use of TLS may potentially disrupt many useful network-based services provided by middleboxes such as content caching, web acceleration, anti-malware scanning and traffic shaping when faced with congestion. As the use of Internet grows to include devices with varying resources and capabilities, and access networks with differing link characteristics, the prevalent two-party TLS model may prove restrictive. We present EFGH, a pluggable TLS extension that allows a trusted third-party to be introduced in the two-party model without affecting the underlying end-to-end security of the channel. The extension stresses the end-to-end trust relationship integrity by allowing selective exposure of the exchanged data to trusted middleboxes.

1. INTRODUCTION

The Internet community has viewed the recent pervasive monitoring efforts as an attack on the Internet [1], [2]. Such large scale pervasive monitoring is indeed an attack, there is simply no arguing that it is not. However, encrypting all traffic as a defence mechanism leads to the preclusion of middleboxes that provide services that benefit networks and users.

Encryption on the Internet is performed by the Transport Layer Security (TLS, [3]) protocol. TLS, and its predecessor, Secure Socket Layer (SSL) were developed at a time in the trajectory of the Internet where electronic commerce was the primary application that required end-to-end security between a browser and a web server. Middleboxes, when they existed at this time, were mostly network/port translators. Today's Internet is much different. It is characterised by plurality of end-user devices, access networks, and middleboxes that apply value-added services (VAS) to the traffic transiting through them. In managed networks, middleboxes provide a variety of services such as parental filtering, caching, accelerating content across slow access links, anti-malware scanning and traffic shaping to avoid congestion; the effect of using TLS is to prohibit such legitimate uses of middleboxes.

The absence of a well-understood and standard manner by which to introduce a middlebox in the two-party TLS model leads to dangerous point solutions like TLS interception proxies that have the effect of destroying end-to-end security completely. Furthermore, thin clients (tablets, smart phones) and the prevalence of computing resources in the cloud are engendering a new architecture, the split browser [4]. Split browsers offload network-intensive computation to servers in the cloud, leaving the thin client to render content more efficiently. By their definition, split browsers introduce HTTP proxy middleboxes between the client and the origin server.

Contributions: It is important that the end-to-end nature of TLS is preserved while accounting for the complexities of the currently deployed Internet. To do so we propose End-to-end Fine Grained HTTP (EFGH) security, a pluggable extension to the TLS protocol that allows middleboxes to be explicitly introduced in the client-server path without affecting the security guarantees of the end-to-end channel. The trusted middlebox is granted access to a subset of the traffic that the principals have agreed upon. The capability is negotiated between the principals through the TLS handshake extension mechanism. Upon completion of the handshake, and if both parties support EFGH, a one round-trip message exchange is required for key establishment between the client, the server and the proxy. EFGH strongly stresses the end-to-end trust relationship integrity and allows the client and server to selectively expose exchanged traffic to trusted intermediaries via a modified TLS record format.

The rest of this paper is structured as follows: Section 2 puts our work in context with existing literature. Section 3 describes the protocol building blocks; Section 4 discusses its security properties. We conclude in Section 5.

2. RELATED WORK

In the absence of techniques like EFGH that allow moderated access of the traffic to intermediaries, enterprises and service providers often resort to questionable solutions that have the adverse and detrimental effect of degrading the overall security. TLS interception proxies [5] are a good example; such proxies straddle two separate sessions and act as a man-in-the-middle (MitM) to decrypt and re-encrypt the content as it traverses through them. The user is not

HotMiddlebox'15, August 17-21, 2015, London, United Kingdom
 ISBN 978-1-4503-3540-9/15/08…$15.00
DOI: http://dx.doi.org/10.1145/2785989.2785990

notified and cannot give consent to the presence of such interception proxies; thus they erode any expectation the user has of the connection being end-to-end secure. In addition, they carry other risks: an organisation may be open to legal exposure as a result of inspecting communications that are intended to be private and such solutions act as an attack target themselves since they are a single point where encrypted sessions are decrypted and available as plaintext.

Loreto et al. [6] improve on transparent interception proxies by allowing a client (browser) to discover and authenticate a trusted proxy and for the user to explicitly provide consent for traffic to flow through that proxy. Unlike our approach, once the trusted proxy has been identified and consented to, it (the proxy) has cleartext access to the information flowing between the client and the server. Loreto et al. further constrain the trusted proxy such that URIs that are available over the *https* scheme do not traverse the proxy. This has the effect of precluding the proxy from performing services that may be of benefit to the user. Peon [7] allows clients to use explicit proxies as well but encourages the use of secure hashes to detect if the proxy applied any transforms to the message. However, unlike our work, it describes a binary proposition for security: either the proxy is trusted and has access to decryption keying material or it is not and traffic through it is encrypted end-to-end. McGrew et al. [8] introduce a TLS extension to allow a chain of TLS proxies to inform a client about the capabilities of the (proxied) server, so that the client could make knowledgeable access control decisions about the server as if the proxies were absent. The main problem with this proposal is that it breaks the TLS end-to-end security assumption by allowing the proxies to actually be MitM entities.

Moving down from application to the transport layer, tcpcrypt is a transport layer encryption protocol [9, 10] that cryptographically protects TCP segments. While tcpcrypt is a good solution for opportunistic encryption, it does not replace TLS. Some drawbacks of tcpcrypt are its use of short-lived keys to provide some forward secrecy and the lack of a key confirmation step in its 4-way handshake. Further down at the IP layer, Kasera et al. [11] and Zhang et al. [12] describe allowing trusted middleboxes in an IPsec association between two endpoints. In Kasera et al. [11] the part of the message that is destined for the proxy is sent in cleartext. Zhang et al. [12] propose dividing the payload into multiple zones, each encrypted with a different key. The primary drawbacks for an IPsec-based solution remain the need for an separate key management scheme, the cost of incurring double encryption when TLS is used on top of an IPsec channel, and its primary use in managed networks to establish secure tunnels between a host and a corporate security gateway.

3. EFGH PROTOCOL

EFGH is made of four basic building blocks:

- The EFGH TLS extension (Section 3.1) allows the two principals to introduce the trusted third party and negotiate a disclosure policy;
- the three-party key exchange protocol (Section 3.2) provides the mechanisms to set up the group security association;
- the framing layer defines how the data is structured to cater for the different visibility (Section 3.3); and lastly,

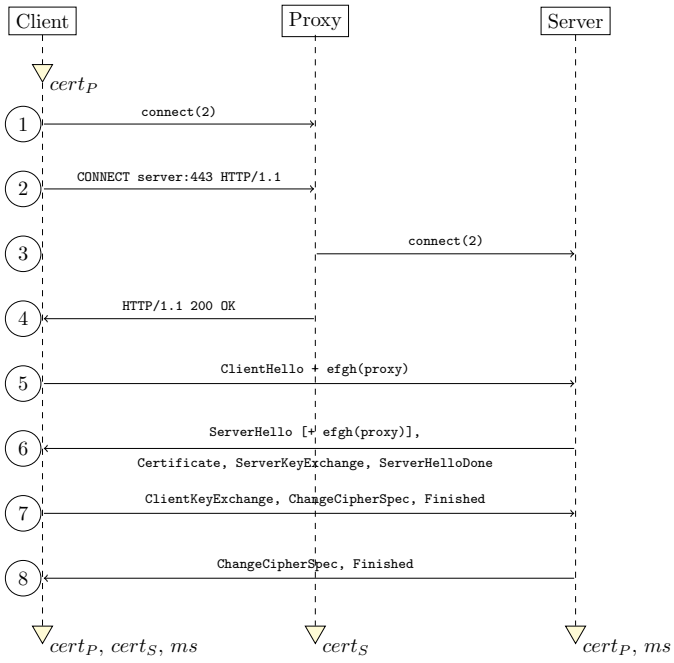

Figure 1: DHE TLS handshake with EFGH option

- the policy language (Section 3.4) determines which subset of the application protocol traffic needs to be disclosed to the third party.

3.1 EFGH TLS extension

EFGH does not modify the normal TLS handshake; instead it moves the necessary three party key negotiation protocol into a separate construct with a well defined interface to the TLS handshake (see Section 3.2). There are two main reasons for this: first, we want to simplify as much as possible the formal analysis required to prove (or disprove) the security of EFGH by building on top of the extensive literature that has been produced regarding the security of the TLS handshake [13], [14]. Secondly, we want to allow independent evolution of the two mechanisms, especially in these early stages of development.

Therefore, the only modification required by EFGH consists of a new TLS extension [15] sent by the client along with the `ClientHello`, which is intended to *introduce* the identity of the proxy to the server, together with the associated fine-grained disclosure policy. The server can decide to accept the presence of the proxy by mirroring the extension, or to deny it, in which case the session falls back to usual two-party TLS.

The basic handshake through a proxy is shown in Figure 1: a client first establishes a TCP connection with a proxy (label 1); the discovery of the proxy (and the services it offers) could be through the proxy auto-config file, DHCP, or by manually configuring the address of a proxy in browser chrome. A HTTP `CONNECT` method is sent to the proxy (2), which opens up a TCP connection to the server identified by the `Host` header (3). Upon the receipt of a `200 OK` response (4), the client upgrades the existing TCP connection through the proxy to TLS (4). The client sends a custom extension (`efgh`) containing the public key (PK) certificate

of the proxy and the fine-grained disclosure policy. If the server also supports the extension, and agrees on the presence of the proxy and the disclosure policy, it echoes the extension in the `ServerHello` message (5). If the server does not support the extension, it simply does not echo it in the `ServerHello` message (6), and the TLS handshake proceeds as if the `efgh` extension was not present at all. The client and server finish their respective handshake messages (7 and 8).

At the end of the handshake, the client knows whether the server supports EFGH or it does not; on its end, if the server supports EFGH, it knows the certified public key (PK) of the proxy. Because the proxy sees all messages starting with the `ClientHello` (4), it also knows whether the client and server support EFGH. Note that up to this point the proxy is a completely passive entity, and is unable to alter the establishment of the end-to-end security association without being detected by the two principals.

3.2 Key exchange protocol

3.2.1 Requirements

The primary goal of the EFGH key exchange protocol (KEX) is to establish the cryptographic keys that are used by the three participants to protect subsequent traffic, and to do so in a way that doesn't compromise the end-to-end security association between client and server created by the *parent* TLS handshake. Along with that, it has been designed to satisfy two additional objectives:

1. no additional round trips (+0-RTT); and
2. same perfect forward secrecy (PFS) characteristics as the parent TLS handshake.

Since the EFGH protocol is initiated by the server, its first two messages (server to proxy, and proxy to client) are piggy-backed on the `Finished` message that closes the TLS handshake. Note that these two messages are completely distinct from the TLS Finished message, and they receive independent cryptographic protection. The remaining two messages are sent together with the first record containing application data from client to server. This way the "+0-RTT" requirement is met. The "same PFS" goal is achieved by deriving the group and end-to-end keys (G and K_{CS}, respectively) from the TLS master secret, thus inheriting its PFS-ness. The remaining keys (K_{CP} and K_{PS}), which serve proxy's origin authentication, are negotiated on the fly via the two Diffie-Hellman (D-H) sub-exchanges – as further explained in the following sections – which always provide PFS.

3.2.2 Protocol Overview

EFGH KEX is a three-party key agreement & distribution protocol that needs a total of four messages to complete. It is logically chained with a *parent* TLS handshake (Section 3.1) which is used to create the *fresh* end-to-end security association between client and server (i.e. ms, the master secret), and to pre-position the proxy and server public keys needed for subsequent signature verifications.

From a high level perspective it is composed of two distinct D-H exchanges both involving the proxy on one side and one of client or server on the other. A further layer dealing with the secure transport of the group key (from client to proxy), and proof-of-possession (from server to client via proxy, and then from proxy to server) is run on top of the two D-H exchanges. All messages have explicit (via certificates) or implicit (via the master secret) authentication.

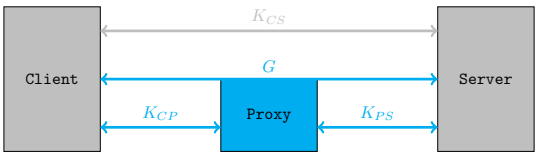

Figure 2: EFGH actors & keys

A successful run of the protocol produces the following keys (Figure 2):

G: shared by client, proxy and server – used to encrypt traffic with group visibility;

K_{CS}: shared by client and server – used to protect end-to-end traffic;

K_{CP}: shared by proxy and client – used to authenticate traffic that is originated by the proxy, e.g. a HTTP response from cache;

K_{PS}: shared by proxy and server – used to authenticate traffic that is originated by the proxy, e.g. sequence number reset due to an end-to-end exchange being short-circuited by the proxy (cache hit, filtering policy match, etc.);

3.2.3 Protocol Steps

The computations and message exchanges are outlined in Figure 3 and discussed next. (In the following, the $\texttt{Sign}_k(\cdot)$ notation represents the signature over the *whole* protocol message using k.)

Step a. The parent TLS handshake seeds the two principals with ms, which is extracted on both sides from the underlying TLS session context using TLS Keying Material Exporters [16]. The same TLS session also securely delivers the agreed-upon proxy public key certificate $cert_P$ to TLS principals, and the server's certificate $cert_S$ to client. Proxy is insecurely informed of $cert_S$: at this stage, and until EFGH completes, proxy can't tell whether $cert_S$ is genuine or not. The proxy and server public key certificates ($cert_P$ and $cert_S$, associated with private keys p and s, respectively) are also shared by all the participants.

Step b. Server derives the group key G from the master secret ms, and computes $\texttt{Tag}_G$, which proves the possession of G and will bind player's instances and messages. It then computes fresh D-H keys and starts the EFGH KEX by sending the first message to proxy which consists of the D-H public key (and parameters) and the proof-of-possession of G, signed using server's private key s. The message is received by proxy, and verified using $cert_S$.

Step c. Proxy computes fresh D-H keys and sends out the second message to client, which consists of its D-H public key (and parameters) together with the signature computed on the D-H and $\texttt{Tag}_G$ received from the server. This signature binds together the proxy identity with the session (and hence with the server), as it includes $\texttt{Tag}_G$, derived from ms.

Client computes G the same as server did in Step b. and subsequently verifies that the signature on the message is valid and that it matches G. If successful, a new D-H private key is generated, and the proxy origin authentication key K_{CP} is derived from the shared D-H key. Client sends a

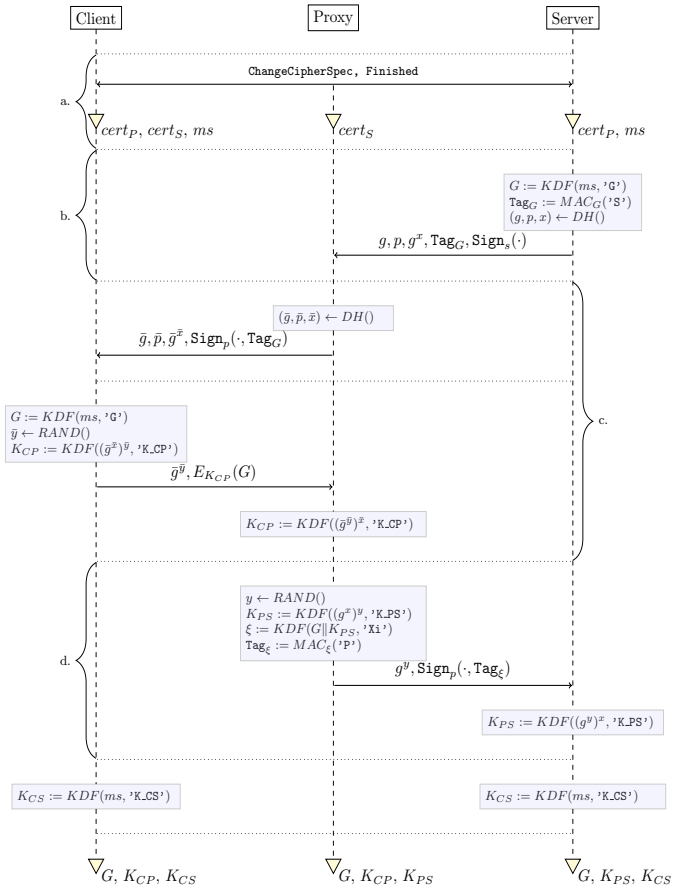

Figure 3: EFGH key agreement

message to proxy which consists of its public D-H key and the group key G encrypted using the K_{CP} itself.

Proxy receives the message, derives K_{CP} and uses it to decrypt G and to verify $\texttt{Tag}_G$, which had been received by server in Step b. At this stage, proxy knows that it shares key G with server and another party (i.e. the client of the parent TLS handshake), and that it shares a K_{CP} with another party (still, the client of the TLS handshake).

Step d. Proxy generates a new D-H private key, and derives its (other) origin authentication key K_{PS} from the D-H public key sent by server in Step b. Then, an ephemeral key ξ is derived by combining G and K_{PS} contributions, and $\texttt{Tag}_\xi$ is computed. $\texttt{Tag}_\xi$ serves as a proof-of-possession of both G and K_{PS} by the proxy. Proxy sends its D-H public key to server and signs it along with $\texttt{Tag}_\xi$. This signature binds together the proxy identity with the session (and hence with the server and client), as it includes $\texttt{Tag}_\xi$, derived from ms, G and K_{PS}. At this stage, proxy possibly shares K_{PS} with server.

Server receives the message, derives K_{PS} and verifies both proxy signature and proof-of-possession over G and K_{PS}. At this stage server knows that proxy knows G, and that it shares K_{PS} with proxy. It also knows that the only entity that could have sent G to the proxy is the client.

Output All keys in Figure 2 have been computed, including the end-to-end key, derived from the master secret independently from any message exchange by client and server.

3.3 Framing Layer

EFGH reuses the TLS framing format – in fact, a TLS Record Layer Header is prepended to each EFGH frame – and defines three new classes of messages: Handshake, Application Data, and Alert (detailed below). These three message classes are completely separate from their TLS counterparts, each being assigned a new `ContentType` from the (currently) unassigned values in the TLS ContentType Registry [17].

Making EFGH framing compatible with TLS is an explicit design choice with a threefold aim. The first is to improve protocol robustness by reducing the chances that other in-path middleboxes modify or strip out EFGH bytes [18]. The second goal is to reuse as much existing software (OpenSSL) as possible while implementing the protocol so that it works without modification with the necessary packet-capture tools (Wireshark, tcpdump). Finally, TLS compatibility allows us to migrate EFGH from an external to a fully in-stack TLS feature in the future with minimal disruption.

Handshake: There are four Handshake frames, each corresponding to one of the key exchange protocol messages defined in Section 3.2. Each has its own format and subidentifier. No additional cryptographic transforms are applied to these messages other than those that they themselves define.

Application Data: Application Data (AD) frames carry end-to-end traffic generated by the application. There are two types of AD frames: those visible to the proxy, and those that are not. Regardless of its type, an AD frame can be broken down in three different sections:

- A common header (H), which carries an explicit sequence number and a flag indicating whether this frame's data are visible to the proxy or not;
- A metadata block (M), which is always made visible to the proxy, containing high level information about the transported data – at a minimum the application protocol, plus any information that can relate this frame to other frames pertaining to the same logical stream (e.g. an unique transaction identifier);
- The data block (D), which carries a specific fragment of the application data.

The blocks are treated differently depending on the overall visibility/opacity of the AD frame they belong to; however, the same Authenticated Encryption with Associated Data (AEAD [19]) algorithm, $E_k(nonce, P, A)$, is used in both cases. Figures 4 and 5 illustrate the applied cryptographic protection: a solid color means the enclosed data is encrypted using the associated key, whereas a single line means the surrounded data is authenticated using the associated key. To guarantee the confidentiality, authenticity and integrity properties of the AEAD-based construction, the same *nonces* must not be used with the same key and distinct plain-text values.

Proxy-visible frames. First, G is used to authenticate and encrypt the metadata and data blocks (i.e. $P := M \| D$), and to also authenticate the header (i.e. $A := H$). Then, the sender uses its key (typically K_{CS}) to authenticate the output of the previous step (i.e. $A := E_G(nonce_1, M \| D, H)$ and $P := \emptyset$):

$$C_G \leftarrow E_G(nonce_1, M \| D, H)$$
$$C_{K_{CS}} \leftarrow E_{K_{CS}}(nonce_2, \emptyset, C_G)$$

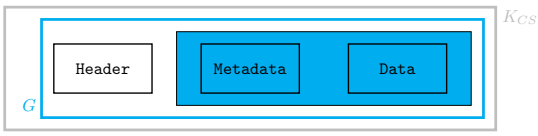

Figure 4: proxy-visible frames

Proxy-opaque frames. First, G is used to authenticate and encrypt M, and authenticate H. Then, the end-to-end key K_{CS} is used to authenticate and encrypt D, and also authenticate the output of the previous step:

$$C_G \leftarrow E_G(nonce_1, M, H)$$
$$C_{K_{CS}} \leftarrow E_{K_{CS}}(nonce_2, D, C_G)$$

In both cases the resulting frame is:

$$nonce_1 \| nonce_2 \| H \| C_G \| C_{K_{CS}}$$

Note that the header block contains the clear-text flag that

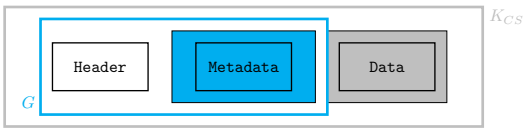

Figure 5: proxy-opaque frames

allows the receiver to decide how to interpret the rest of the frame.

Alerts: Alerts are used for in-band signalling of protocol errors/warnings (similarly to TLS), but also for EFGH specific events like a sequence number re-synchro-nisation. The latter is needed whenever a short-circuited response is sent from proxy instead of being forwarded to the intended destination (typical use cases include a cache hit on a transparent caching proxy, or a filter match on a filtering proxy). Alerts are encrypted (with G) or sent in clear-text, depending on the specific protocol phase in which they are emitted.

3.4 Policies

An EFGH policy encodes the fine-grained disclosure rules. In its current form it is a simple two column matrix: <*allowed party*, *protocol element tag*> that informs the principals about *who* is allowed to see *what*. In particular, *allowed party* is one of "3rd-party" or "anyone", while *protocol element tag* is an unique identifier defined by the application protocol that is encapsulated by EFGH. For HTTP, a tag namespace could be organized as follows:

- *http.request-line*
- *http.response-line*
- *http.header.content-type*
- *http.header.user-agent*
- *http.body[start-pos, end-pos]*
- ...

The enforcement of the policy rules is done by the sending party, which starts by denying 3rd-party visibility of anything, and then discloses the protocol element listed in the policy. Examples: to allow a parental-control service based on URL filtering: <*3rd-party*, *http.request-line*>; to allow a traffic prioritisation based on content-type: <*3rd-party*, *http.header.content-type*>.

4. KEY EXCHANGE SECURITY

We designed our protocol to be as modular as possible in part to be able to provide a simple security argument. We assume a successful completion of TLS, which securely provides the TLS client (C) and server (S) with $cert_P$ and $cert_S$ and a shared key ms. In EFGH, each player derives its new keys from authenticated DH material and is also bound to the preceding TLS session key material. A more detailed security argument and the explanation of protocol messages and security guarantees they provide is done in step-by-step discussion of Section 3.2.3. Here we just reiterate that tags derived from ms and players' credentials are used to securely bind together all messages and player identities. For example, the DH exchange between C and proxy (P) is authenticated by both players' public keys, as well as bound to ms via $\texttt{Tag}_G$. Similarly, the DH exchange between P and S is authenticated by their respective public keys, and is bound to the same ms via $\texttt{Tag}_\xi$.

5. CONCLUSIONS AND FURTHER WORK

We have presented a model that allows two TLS principals to explicitly introduce an in-path, third party in an otherwise end-to-end secure session, and to expose to the third party a well defined and explicitly agreed subset of the exchanged data. The proposed solution is expected to have negligible impact on latency (in particular time to first byte) and on additional computational resources when compared to vanilla TLS. We construct our solution on top of TLS, thereby inheriting the security properties of TLS. Future work will study the computational overhead of the handshake and the differential framing; extend the handshake to more than one intermediary; and explore the feasibility of reusing EFGH to encapsulate protocols other than HTTP.

6. REFERENCES

[1] S. Farrell and H. Tschofenig. Pervasive monitoring is an attack. RFC 7258, May 2014.

[2] Internet Architecture Board. IAB statement on Internet confidentiality. https://www.iab.org/2014/11/14/, 2014.

[3] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, August 2008.

[4] Angeliki Zavou, Elias Athanasopoulos, et al. Exploiting split browsers for efficiently protecting user data. In *Proceedings of the 2012 ACM Workshop on Cloud Computing Security Workshop*, CCSW '12, pages 37–42, New York, NY, USA, 2012. ACM.

[5] Jeff Jarmoc and Dell SecureWorks Counter Threat Unit. SSL/TLS interception proxies and transitive trust. *Black Hat Europe*, 2012.

[6] Salvatore Loreto, John Mattsson, et al. Explicitly authenticated proxy in HTTP/2.0. IETF Internet-Draft (work-in-progress), July 2014.

[7] Roberto Peon. Explicit proxies for HTTP/2.0. IETF Internet-Draft (work-in-progress), June 2012.

[8] David A. McGrew, Dan Wing, et al. TLS Proxy Server Extension. IETF Internet-Draft, July 2012.

[9] Andrea Bittau, Michael Hamburg, et al. The case for ubiquitous transport-level encryption. In *in 19th Usenix Security Symposium*, August 2013.

[10] Andrea Bittau, Michael Hamburg, et al. Cryptographic protection of TCP streams. IETF Internet-Draft (work-in-progress), July 2014.

[11] Sneha Kasera, Semyon Mizikovsky, et al. On securely enabling intermediary-based services and performance enhancements for wireless mobile users. In *Workshop on Wireless Security, 2003*, pages 61–68, 2003.

[12] Yongguang Zhang and Bikramjit Singh. A multi-layer IPsec protcol. In *in 9th Usenix Security Symposium*, August 2000.

[13] Karthikeyan Bhargavan, Cèdric Fournet, et al. Proving the TLS Handshake Secure (As It Is). In *Advances in Cryptology, CRYPTO 2014*, volume 8617 of *Lecture Notes in Computer Science*, pages 235–255. Springer Berlin Heidelberg, 2014.

[14] Hugo Krawczyk, Kenneth Paterson, et al. On the Security of the TLS Protocol: A Systematic Analysis. In *Advances in Cryptology, CRYPTO 2013*, volume 8042 of *Lecture Notes in Computer Science*, pages 429–448. Springer Berlin Heidelberg, 2013.

[15] D. Eastlake 3rd. Transport Layer Security (TLS) Extensions: Extension Definitions. RFC 6066, January 2011.

[16] E. Rescorla. Keying Material Exporters for Transport Layer Security (TLS). RFC 5705, March 2010.

[17] Internet Assigned Numbers Authority. Transport Layer Security (TLS) Parameters, 2015.

[18] Jim Roskind. QUIC, Quick UDP Internet Connections. https://goo.gl/XMfO6Q, 2013.

[19] Phillip Rogaway. Authenticated-encryption with associated-data. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 98–107. ACM, 2002.