Network Working Group
Internet-Draft
Intended status: Experimental
Expires: May 3, 2018

H. Tschofenig
ARM Limited
M. Baugher
Consultant

October 30, 2017

Layered DTLS/TLS
draft-tschofenig-layered-tls-00

Abstract

TLS and increasingly also DTLS are frequently used to provide channel security for Internet of Things (IoT) communication. On the Web and smart phones, TLS is already the defacto approach for securing protocol interactions. While the end-to-end security offered by TLS, particularly TLS 1.3, is already too much for some, there are others who believe that TLS is insufficient. While the former group is working on ways to weaken TLS security, the latter group is interested in designing an application layer security solution. Whether application-layer security is used in addition to or as a substitute for transport-layer security is of secondary importance. However, the security needs for such an application layer solution are similar, if not identical, to those that drove the design of TLS. This is for an obvious reason: Security requirements are not tied to the name of a security protocol nor to the layer at which it is executed. One can make this observation also in other areas, such as with the increasing similarity of Internet Key Exchange (IKE) and the TLS handshake protocols.

These discussions within the IETF inspired the document authors to explore whether TLS could actually be used also at the application layer and how complex it would be. We call this approach "Layered TLS" since TLS may, in some scenarios, be executed at two layers: above the transport layer in the traditional manner and also at the application layer.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Copyright Notice

Table of Contents

1. Introduction

"Layered TLS" addresses two problems with TLS connections to middle boxes: End points cannot authenticate each other, and each intermediary gains more access to decrypted messages than it needs. The lack of end-to-end authentication puts a service at risk of message-forgery attacks. Services lacking end-to-end encryption, moreover, leak information and violate the principle of "least privilege." End-systems that use TLS or DTLS may benefit from applying TLS to secure application messages, end-to-end, and reusing TLS infrastructure and open-source software. This short report solicits ideas and opinions from the Internet standards community on using TLS at the application layer for message security, end-to-end.

In this document, "message security" means sender authentication, message-integrity protection, and confidentiality of at least parts of the message. "End-to-end authentication" means that a network receiver can determine the sender of a message - and vice versa. And "end-to-end confidentiality" means that the receiver and only the receiver can read the sender's message. The need for these security services on the Internet is well established and defined [RFC4949]. Nonetheless, most Internet services today lack end-to-end security, which is the subject of this work.

Internet mail endpoints, for example, are periodically offline and need always-on SNMP servers to store mail. StartTLS is a standard that authenticates mail servers and encrypts messages on the network, but transport-layer security cannot cryptographically authenticate mail endpoints or encrypt messages between them. That requires PGP, S/MIME or some other method of message authentication and encryption. The absence of this layer of email security has caused enormous problems for people, organizations and nations worldwide, but insecure practices persist.

The same had been generally true for chat and VoIP services prior to the introduction of application layer security protocols: XMPP deployments, for example, have generally failed to protect message

contents using S/MIME, as specified by XMPP standards. Instead, early deployments used only transport security between chat servers, similar to StartTLS between SNMP servers. Chat, however, has evolved a succession of application-layer security protocols such as Off-the-Record and the "double-ratchet" key-management schemes found in Signal/Noise and its predecessors. VoIP endpoints have initially passed keys in a hop-by-hop fashion between SIP proxies (with SIP proxies seeing the keys in plaintext) before a range of other end-to-end security solutions were developed, including DTLS-SRTP. The emergence of end-to-end security for VoIP and chat is a bright spot for improving privacy and security through middle boxes.

Internet of Things (IoT) communications have a similar problem: IoT services commonly use middle boxes for connecting different IoT islands. Unlike most chat deployments, however, IoT devices have additional constraints: they are typically limited in memory, storage, processing power and network capacity. This complicates the problem of securing these applications and raises the specter of many more new and untested security protocols, including problematic key management, seeZillner [1] and authorization systems, see SmartThings [2].

It is unlikely that a single security protocol can satisfy all IoT applications since some of them will require one-shot messages, like firmware updates. But CoAP, MQTT, HTTP and other standards increasingly use DTLS or TLS for communications security in at least some constrained environments. Ideally, deployments should avoid the use of middle boxes to simplify the overall architecture. However, for those cases where this is not feasible this work seeks to authenticate and encrypt through middle boxes by reusing TLS [I-D.ietf-tls-tls13], [I-D.ietf-tls-dtls13] infrastructure and open-source software implementations.

This is a work in progress to develop an architecture, design and implementations for layered TLS services in both the end system and middle box. Some problems and evaluation metrics are also considered in the summary.

2. Terminology

"Layered TLS" can refer to a TLS or DTLS association between endpoints. Also, as used in this document, "middle box" is synonomous with application-layer gateway, hub, semantic gateway, gateway or proxy.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

3. Layered DTLS Use Cases

Two use cases are considered. Both of these examples use DTLS.

3.1. Application-Layer DTLS in the end device

It is well understood that a TLS or DTLS function is relocatable from an end system to a middle box, and this is true of layered TLS: The end-device might use application or transport security or both. This is necessary given the diversity of today's IoT products. For example, One product runs TLS from a gateway to a user device, which it also connects to a ZigBee device, see Traedfri [3]. The gateway in Figure 1 is a "semantic gateway" that converts one set of operations to another.

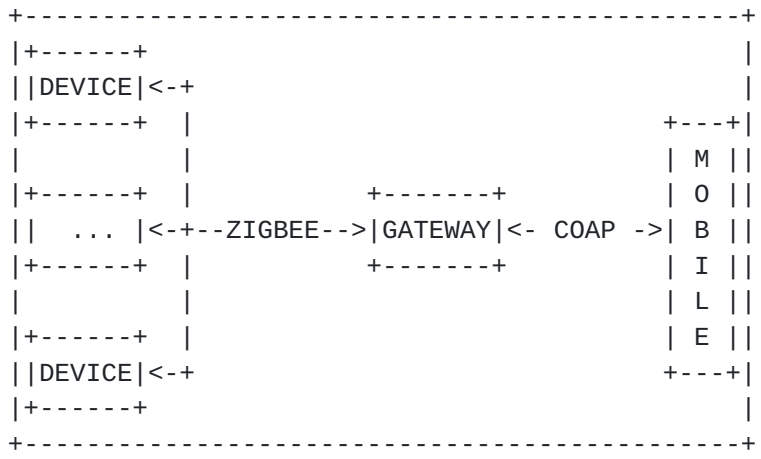

Figure 1: A middle-box semantic gateway

In Figure 1, DTLS secures CoAP messages en route. An additional DTLS layer would improve end-to-end security for data that are sent over the ZigBee network but intended only for the mobile endpoint and not needed by the GATEWAY function.

3.2. Transport-layer DTLS to the end device

In another topology, CoAP is run between a constrained end-device and a cloud server, aggregator, or some type of proxy. In this case, some non-constrained protocol rather than CoAP may run on the other side of the middle box. But either way, all device data are accessible to the proxy en route to their legitimate destination, such as the mobile device shown below.

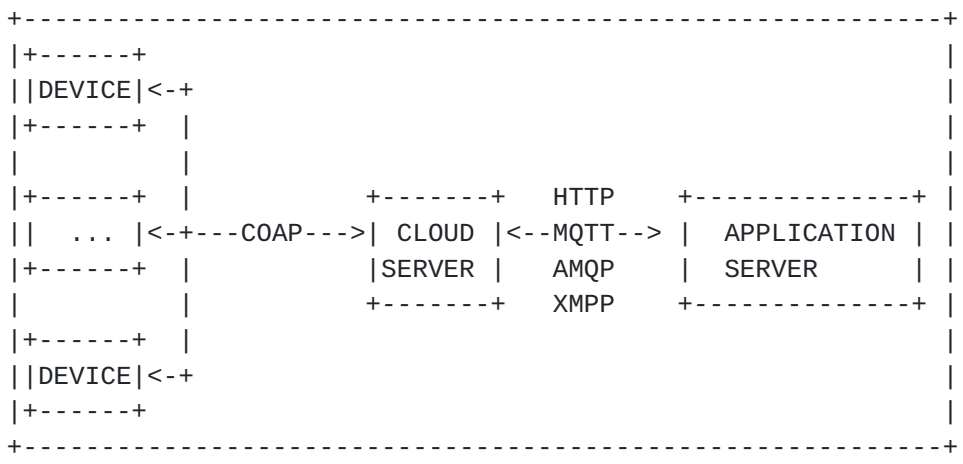

Figure 2: DTLS to the IoT Endpoint

Figure 2 illustrates the problem of relying on hop-by-hop transport security: The end systems have no assurance that all hops run it. Thus, in addition to method exposure inside middle boxes, end systems have no assurance of authentication and confidentiality end-to-end. A partial remedy to this problem is to secure messages between the DEVICE and the APPLICATION SERVER. This is done independently of the Figure 2 CLOUD SERVER. This is the goal of layered TLS.

4. Design Rational

At a first glance, the use of TLS may appear as an overkill since application layer security often gives the perception of being lightweight. However, we argue that the same security services are needed.

When protecting data sent by a device, one obviously might want to offer integrity and confidentiality protection. Additionally, it is important for the recipient to verify that the data was sent by a specific party, a property provided by data origin authentication. Today, ciphers offering authenticated encryption with additional data (AEAD) are the preferred approach.

So far, a simple JOSE or COSE object provides us such security services. When we design such a system we might want to start with public key cryptography to simplify key management. However, public key cryptography has the unfortunate side-effect of being heavyweight, particularly on a per-message basis. Additionally, there is the risk of denial of service attacks when an entity needs to make heavy computations based on a single message only.

The classical design to amortize the cost of public key cryptography is to use two phases: In the first phase, we use public key cryptography for entity authentication. We then exchange symmetric keys for use in the second phase where data is protected. The idea is that the first phase happens less frequently than the second where application data protection takes place. Ideally, we want to avoid the first phase as much as possible, which is why we want to cache symmetric keys for fast re-authentication (or resumption).

Best current practice in cryptography today suggests to also incorporate perfect-forward secrecy in the derivation of symmetric key utilizing a Diffie-Hellman approach. Additionally, it has long been a requirement to take crypto-agility into account, since cryptographic algorithms age and different communities tend to have different preferences for algorithms.
Finally, a modern protocol design also needs to consider extensions to the main protocol via a negotiation mechanism.

Voila! We have just re-designed TLS (minus all the security analysis). Many of the needed security features in today's security protocols are not bound to the name of the protocol but are rather a side-effect of the security and privacy services needed for modern applications and the best current cryptography design practices.

Every time you hear "TLS is complex" or "TLS is so heavyweight" ask yourself whether this "complexity" or "weight" is the result of TLS/DTLS or rather the result of security services that a specific application needs.

5. Layered TLS Design

In a communications protocol, a "layer" is often associated with a communications-protocol header, which conveys addressing or other "metadata." Message metadata are considered in one subsection below.

The next subsection considers where headers and TLS layer services are applied along the service path. Figure 2 shows a case where a transport-layer DTLS connection originates at the constrained device. Figure 1 shows a case where the constrained device has its own link-layer security and the GATEWAY acts as the terminus of a DTLS connection. In either case, an application-DTLS layer might be located at the constrained device or at the middle box.

5.1. Relocatable layers

An application developer has to decide whether to use TLS or DTLS at the application layer, and this decision depends on the properties offered by the lower layers, end-to-end. If the lower layer has a

reliable transport service (usually TCP), then TLS is RECOMMENDED. But if the lower layer has an unreliable service like UDP or connects to a middle box, then DTLS is RECOMMENDED. DTLS operates on top of unreliable transports while TLS assumes reliable transports.

By design, TLS layers are independent, but code and data can be shared when two layers share a physical device. More commonly today, the layers can originate either at a communications endpoint or at some middle box in the service path.

5.2. Application-specific layer headers

By design, application-layer TLS secures services through middle boxes. An upper TLS or DTLS layer restricts access to message data inside the middle box. Nonetheless, middle boxes often need some information from and about the message for message processing. This information is referred to as "metadata," or information about the message that may be conveyed in a header, separate from the content of the message and usually unencrypted. [I-D.ietf-core-object-security] does not require that header metadata be moved to an external header, however, in order to reduce overhead in a constrained device.

Application-layer encodings, such as CBOR/COSE or JSON/JOSE define message metadata that need to be shared with middle boxes in two varieties, protected or unprotected, i.e. integrity-protected or re-writable [RFC8152]. The reader should note that the use of the term "layer" in this document is different from the COSE standard in that a TLS layer is a TLS or DTLS connection at or above the transport layer, i.e. source and destination addresses define the layer.

The identification and encoding of message parameters into protected or unprotected headers is very much a function of the particular application service, its content and its use of middle boxes. For example, OSCORE maps message fields into protected, protected and encrypted, or unprotected fields. These fields are from the COSE-encoded object parameters that are needed by CoAP proxies or other types of middle boxes [I-D.ietf-core-object-security].

As important as metadata construction might be, the first solution pursued in this work is to scale down to a very simple design that might prove useful, i.e. improve security. For example, two applications listening on remote ports might not need application-to-application address metadata or other types of service-routing parameters. In this case, there is no need for a header. But in many practical cases do require a header, such as routing through an application overlay, and RESTful endpoints. These cases will be considered more closely in future implementation work.

6. Functional Design

The functional design assumes that an authorization system has established operational keys for authenticating endpoints. In a layered design, this needs to be done for each layer, which may operate in two separate authorization domains.

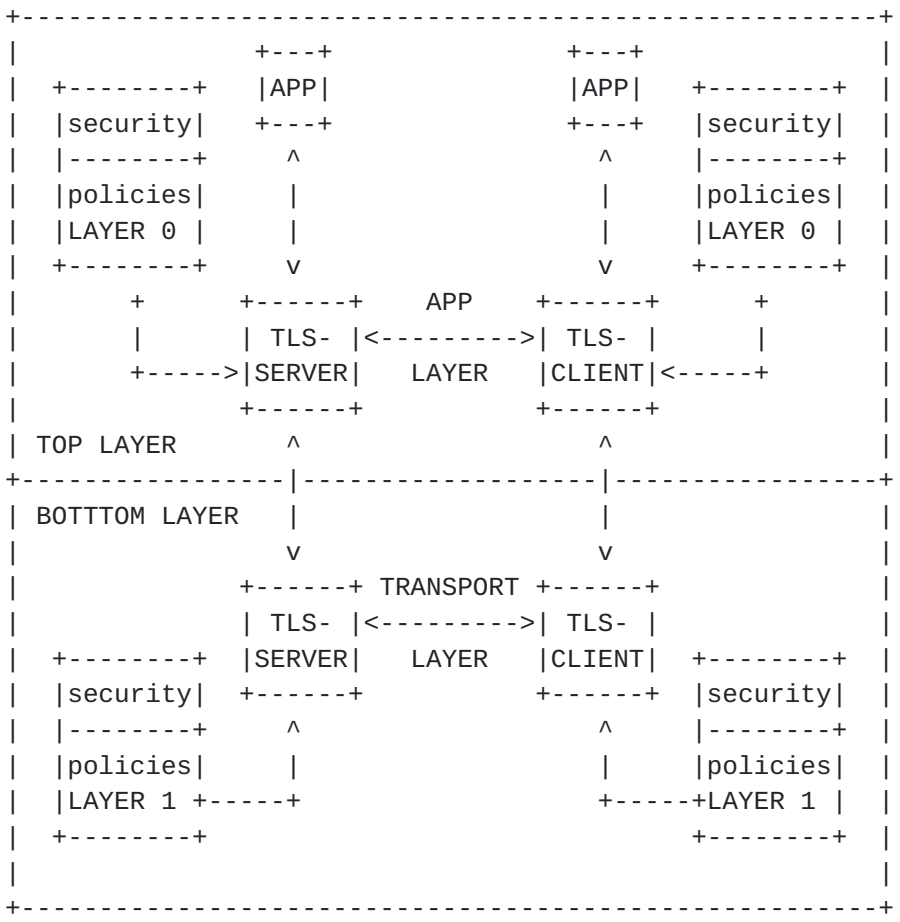

Figure 3

Thus, the security policies of one layer are distinct from those of another in Figure 3. They may overlap, but that's not necessary or perhaps even likely since one layer operates end-to-end, the other hop-by-hop, and the two often have different authorization domains,

TLS can protect IoT device-to-hub communications "on the wire" using the "bottom layer" of Figure 3, and it can protect application data inside the hub/application-gateway using the "top layer." This is needed. Application and transport security each have a role to play. Transport security restricts access to messages on the networks, notably application TLS headers,if any, and application-layer TLS restricts access to messages inside middle boxes.

Thus, Figure 3 accepts an application-layer message, which gets encrypted and integrity protected with optional headers that MAY be integrity protected or not. In the most general case of Figure 3, the resulting TLS message and headers are passed to a TLS socket, which may have a different security policy than the application layer client_hello that the app sends in response to the first message. This application message triggers the transport layer to send its own client_hello to the TLS server or endpoint, which is often a middle box.

7. Summary

The use of TLS/DTLS for protection of payloads at the application layer does not require much or any standardization. However, conceptually it is a big step. With the capabilities offered by TLS/DTLS a wide range of security services are available to the application developer.

Future work will compare the protocol, message, and platform resource demands of layered TLS with given sets of policies and workloads. The layered TLS design presented here is applicable to TLS 1.3 and earlier versions. But TLS 1.3 is chosen for the Layered TLS prototype, however, to make use of zero-RTT and other new features, as well as ongoing improvements anticipated in future revisions of TLS and DTLS 1.3 specifications [I-D.ietf-tls-tls13][I-D.ietf-tls-dtls13].

The authors are still undecided whether Layered TLS should establish keys for use with the TLS/DTLS record layer only or should establish keys for use with COSE/OSCoAP as well.

8. Security Considerations

Layered TLS is intended to improve security for an Internet service by offering application layer security.

In general, adding more security code increases complexity and can thereby make the service less secure. However, in this case the solution re-uses already existing code and utilizes it twice, at different layers.

In any case, it is RECOMMENDED that the layers be isolated and the line between the layers in Figure 4 effectively firewalls one layer off from another using platform features that will foil cross-layer attacks in platforms with two layers.

9. IANA Considerations

There are no IANA Considerations in this draft.

10. References

10.1. Normative References

[I-D.ietf-tls-dtls13]
Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", draft-ietf-tls-dtls13-02 (work in progress), October 2017.

[I-D.ietf-tls-tls13]
Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-21 (work in progress), July 2017.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

10.2. Informative References

[I-D.ietf-core-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", draft-ietf-core-object-security-06 (work in progress), October 2017.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <https://www.rfc-editor.org/info/rfc4949>.

[RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <https://www.rfc-editor.org/info/rfc8152>.

10.3. URIs

[1] https://www.blackhat.com/docs/us-15/materials/us-15-Zillner-ZigBee-Exploited-The-Good-The-Bad-And-The-Ugly-wp.pdf

[2] https://web.eecs.umich.edu/~earlence/assets/papers/smartthings_sp16.pdf

[3] http://www.sensorsiot.org/145-ikea-tradfri-hack-with-gateway/

[4] https://thekerneldiaries.com/2016/06/13/openssl-ssltls-within-a-different-protocol/

[5] https://wiki.openssl.org/index.php/EVP_Authenticated_Encryption_and_Decryption

[6] https://tls.mbed.org/kb/how-to/mbedtls-tutorial

Appendix A. Implementation

An implementation with two TLS layers has two TLS or DTLS state machines, two sets of policies, and key store. A device that physically hosts two TLS layers results in roughly twice as much state and twice as much header overhead, than a single layer. This is not unusual.

Implementation work is ongoing on both OpenSSL and mbedTLS libraries.

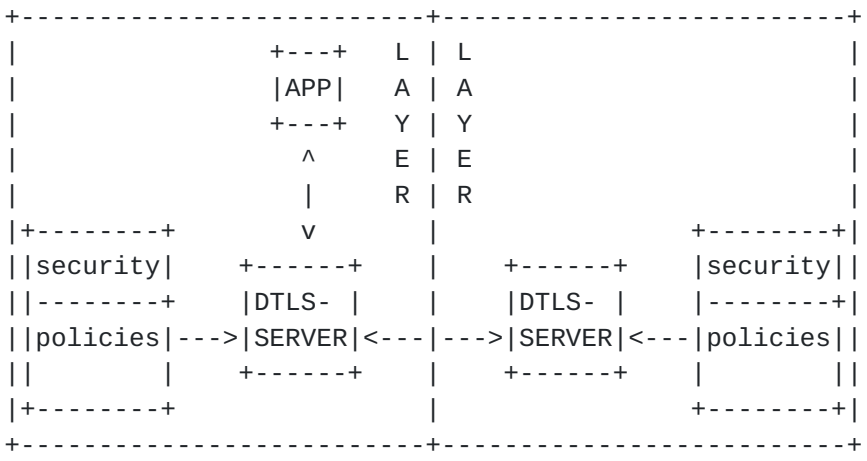

Figure 4: Prototype Functional Block Diagram

A.1. OpenSSL

There is documentation online [Norrell] that describes a way to use the OpenSSL library to create a protocol-independent TLS encapsulation of application-level messages including Handshake messages, see Norrel [4]. Normally, the easiest OpenSSL API calls for a TLS session are SSL_write and SSL_read, shown in the next figure from Norrel.

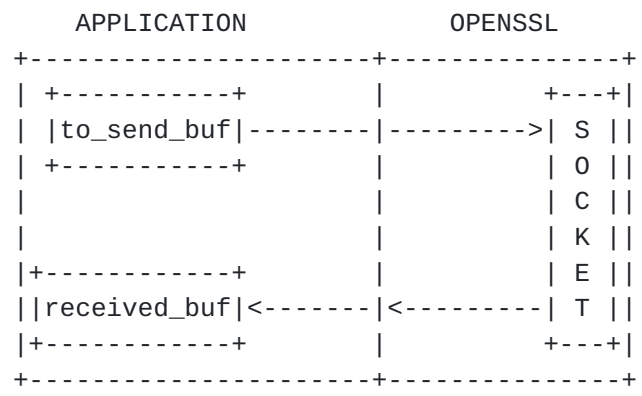

Figure 5: OpenSSL calls send and receive user messages as TLS messages over a socket

As shown above, OpenSSL writes a user buffer to a TLS message and sends the message out a communications socket. The converse is done on the receiving side. What's needed to create a layer independent of the host IP stack is to send the TLS message to a buffer rather than to a communications socket and do the reverse on the receiving side. This is shown in Figure 6, which is also from Norrell..

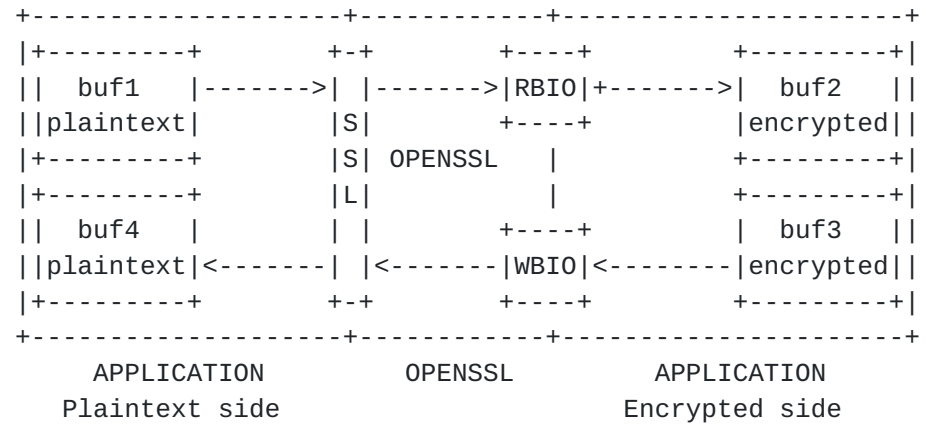

Figure 6: BIO interface provides raw TLS message

In #SSLBIO, the SSL_write and SSL_read calls use an OpenSSL context that is bound to a buffer rather than a socket. To retrieve a TLS message for OpenSSL, the application does a BIO_read of the RBIO buffer on the sender side and the converse on the receiver side.

In between the send and receive operations, message headers are optionally created and added or parsed and removed. The authors of [I-D.ietf-core-object-security] argue that there is a general need for both unprotected headers and protected headers with a message integrity check. OSCORE describes how to map COSE parameters to a message header. OSCORE uses the AEAD transform to take protected-

header data as additional data without replicating the parameters inside the message. The message authentication code is created (on send) and checked (on receipt) in the OpenSSL layer of Figure 6. Prior to calling SSL_write, therefore, the application ("APPLICATION" in #SSLBIO) will add any needed protected and unprotected headers. If a protected header is to be included, the protected-header parameters will be passed as additional data to an AEAD interface, such as a manual encryption using EVP prior to SSL_Write and following SSL_read, see OpenSSL [5]. As mentioned above, protocol-dependent signaling is often needed for those application protocols that support application middle boxes to ensure that Handshake protocol messages are not cached en route. These are added as new parameters in the protected or unprotected headers.

A.2. mbedTLS

Most of our work is focused on adding layer functionality to a TLS 1.3 prototype based on mbedTLS. Like OpenSSL, mbedtls has a straightforward method to receive and send DTLS or TLS-encapsulated data. This uses mbedtls_ssl_set_bio, see mbedtls [6].

Beyond how to use the mbedTLS application API, however, are deeper questions about efficiency, layer isolation, constrained and scalable multi-processor platforms, and platform resource sharing between layers. Our current work extends existing mbedtls example application code to prototype a single layer and multi-layer application program. Early code has been contributed to the mbedtls github repository.

Appendix B. Contributors

The authors wish to thank our friends in the IPSO Alliance for their ideas and criticisms, particularly Per Staehl, Ned Smith, and Alan Grau.

Authors' Addresses

Hannes Tschofenig
ARM Limited

EMail: hannes.tschofenig@gmx.net

Mark Baugher
Consultant

EMail: mark@mbaugher.com