Enhancing Transport Layer Security with Dynamic Identity Verification and Authentication (DIVA)

Maintaining and Enhancing SSL/TLS Reliability

Laurie Perrin
Sequor Systems
Orlando, USA
info@sequorsystems.com

Andre Brisson
Whitenoise Laboratories Canada Inc.
Vancouver, Canada
abrisson@wnlabs.com

***Abstract*—This electronic document outlines an extension to the Transport Layer Security (TLS) protocol for incorporating the enhanced peer authentication and data protection provided by Whitenoise Laboratories' Dynamic Identification Verification and Authentication (DIVA) cryptosystem.**

Keywords—network security; one-time pad; symmetric key; non-arithmetic cipher; pseudo-random; key-based authentication

I. INTRODUCTION

With advances in computing technology, particularly in the field of quantum computing, once infeasible vectors of cyberattack now pose a serious threat to the security of widely deployed mathematical- or arithmetic-based cryptographic algorithms such as those employed within transport layer security (TLS). In particular, successful attacks now exist for National Security Agency Suite B cipher algorithms (including RSA[1] and AES[2]). By enhancing TLS with Whitenoise Laboratories' (WNL) non-arithmetic, key-based cryptographic algorithms, extant and emerging network security vulnerabilities can be eliminated. The TLS extension outlined in this paper specifies a standards-compliant method for strengthening transport security by protecting critical aspects of the TLS protocol from eavesdropping or tampering through use of WNL Dynamic Identity Verification and Authentication (DIVA) technology.

II. TLS-DIVA CONCEPTS

The WNL DIVA cryptosystem[3] implements a set of algorithms which provide the provably secure features of a one-time-pad. In operation a unique, symmetric key (a DIVA key) encapsulates the per-identity security parameters defined by these algorithms. TLS-DIVA enhances network security through use of these symmetric keys in endpoint identification and data encryption and validation.

A DIVA key can be used to create, in a deterministic fashion, an arbitrary quantity of computationally strong pseudo-random data. A (bounded) segment of data developed in this fashion is referred to as a keystream. The successful deciphering of any data enciphered (mixed) with a DIVA keystream requires knowledge of both the specific DIVA key utilized and the precise location at which the keystream was developed and consumed in the encryption process. A central aspect of DIVA security concerns safeguarding the secrecy of deployed keys. A DIVA key is never communicated following its secure deployment.

A. Peer Identity in a TLS-DIVA Session

As per protocol operation, the TLS handshake process entails the exchange of server and, optionally, client digital certificates between connection endpoints. The certificates serve to establish the identity of the presenting party. TLS-DIVA complements this authentication mechanism through use of a previously established secret which is known only by trusted parties (minimally the client and the server). This secret is composed of a DIVA key and a key identifier which uniquely identifies the key within some authoritative domain of employ. Since every TLS-DIVA session negotiates a fresh set of cryptographic parameters, knowledge of this secret conveys dynamic server and client identity validation to the TLS protocol.

B. Enhanced Encryption in a TLS-DIVA Session

TLS encryption may be enhanced by incorporating WNL DIVA at the TLS *ciphersuite* level. By integrating DIVA in this way, successful encryption and decryption of transmitted data requires, in addition to the TLS cipher key, possession of the particular DIVA key as well. This requirement conveys the important advantage that any compromise in the TLS session key (or e.g. a certificate's corresponding private key) does not, on its own, permit successful decryption of the TLS session's application data.

C. TLS-DIVA Modes of Operation

In principle, employing a DIVA keystream as a one-time pad guarantees forward secrecy against any compromise of consumed keystream data. To this end, typical DIVA operation proceeds exclusively on the part of authorized peers, with sequential consumption of some session-specific keystream.

In this mode of operation, the keystream position at which data is produced and consumed is known implicitly by both

connection peers. Thus, a successful participant in a DIVA-secured connection must possess both the identifying DIVA key and the position of the corresponding keystream from which DIVA encryption proceeds.

TLS-DIVA supports the above mode of operation, for scenarios in which a single connection exists between a specific client and a specific server. However, network topologies potentially associate many servers with a given client. In such cases, centralizing and serializing DIVA keystream consumption may prove inefficient or otherwise problematic. Having a multiplicity of servers, each configured to define independent, session-specific keystreams for a given client identity–that is, for a particular DIVA key–is highly advantageous.

The following approach to enhancing TLS with DIVA support offers such an alternate mode of operation, allowing independent and concurrent DIVA keystreams to be generated from a particular DIVA key. In this mode, the server determines and conveys a session's initial keystream position to the client. By delegating this role to the server, the need to coordinate keystream processing on the part of multiple servers is eliminated.

Although keystream space is reasonably on the order of terabytes or greater, whereas such keystream production is not serialized, there remains the *potential* for DIVA key-specific keystreams to overlap. This fact contradicts the one time use security notion of a one-time pad, and for this reason TLS-DIVA adopts additional measures to ensure the security of DIVA operation.

The first measure retains keystream output secrecy by requiring that no DIVA keystream output be mixed with known values, or with data that an attacker might somehow be able to distinguish from randomness. Thus, no attack through statistical analysis can be mounted against such keystream-enciphered data.

In TLS-DIVA, this is accomplished by restricting keystream use to enciphering genuinely random or cryptographically strong pseudo-random protocol data. TLS employs such random data in multiple aspects of the protocol, for instance in composing the 28-octet *random_bytes* sub-field of TLS *ServerHello* handshake message. Another example is the initialization vectors (IVs) factored into certain TLS ciphersuites.

This measure protects against, for example, the scenario whereby an eavesdropper obtains a target server's (e.g. RSA) private key. Owing to DIVA enciphering of the ServerHello random_bytes, the attacker would be unable to decrypt any captured TLS session's master secret and, therefore, unable to decipher the records of the TLS-DIVA protected session.

The second measure safeguards the secrecy of the (server-specified) starting position of a TLS-DIVA session keystream. To prevent a possible eavesdropper from ascertaining what this position ~~is~~ is, before transmission, it is enciphered in such fashion as to permit only a party in possession of the correct DIVA key to obtain it and so successfully complete the TLS handshake.

The procedure of mixing keystream output solely with random protocol data while preserving the secrecy of keystream position allows TLS-DIVA to multiplex DIVA key utilization yet maintain the security offered by a one-time pad.

The implications of this approach warrant some further analysis. In the first case, Chattopadhyay and Zuckerman have shown[6] that it is possible to extract quality random data by combining two or more weak sources of entropy; likewise mixing the pseudo-random data of DIVA keystream output with TLS protocol-level random fields improves the security strength of potentially suspect sources of randomness. In the second case, were the security of TLS algorithms compromised so as to make possible the computation of some portion of a DIVA keystream, the *position* of this compromised portion would remain secret and, therefore, would not constitute a vector for attack upon other TLS-DIVA protected network traffic.

III. Integration with the TLS protocol

In order to foster adoption and ease incorporation into existing systems, changes to the TLS protocol have been kept to a minimum and support for all present and proposed versions of TLS is maintained. The proposed DIVA support for TLS complies with the requirements for TLS extensions[7] and imposes no changes upon either the structure of protocol messages or message flow.

A. TLS-DIVA Requirements

A DIVA supporting TLS implementation must, minimally, perform the following:

- Exchange the client and server hello extensions defined below.
- Encipher the ServerHello random_bytes sub-field with DIVA.

In modifying the TLS handshake, a TLS-DIVA compliant server constructs and records the value of the ServerHello random field as described in TLS, however the value reported to the client is first enciphered by mixing the random_bytes sub-field with DIVA keystream data. The client uses the details provided by the TLS-DIVA extension embedded within the extended ServerHello message to compute identical keystream data with which to decipher the original random_bytes value.

Should both client and server possess identical DIVA key and offset details, the TLS handshake will complete successfully. A discrepancy in the keystream, however, will cause those cryptographic operations involving the ServerHello random_bytes sub-field (notably derivation of the TLS master secret) to disagree and thus cause the handshake to fail.

Note that TLS employs a hash of handshake messages for mutual validation on the part of transport endpoints. This hash is verified by comparing the *verify_data* field of TLS protocol *Finished* messages exchanged by client and server endpoints. Whereas this hash is based upon the handshake messages transferred by both peers, hashing of the ServerHello message

must be done *after* the random_bytes sub-field is enciphered with the DIVA keystream.

B. Employing DIVA with TLS Ciphersuites

Support for DIVA dynamic authentication and continuous encryption within TLS can be achieved by incorporating the DIVA keystream into the encryption and decryption performed by a TLS session's selected ciphersuite.

A ciphersuite belongs to one of three categories: chain-block (CBC), stream, and authenticated-encryption-and-data (AEAD) ciphers. As TLS version 1.2 mandates[8] support for the *TLS_RSA_WITH_AES_128_CBC_SHA* chain-block cipher, support for the chain-block category of ciphersuite is outlined presently.

A TLS chain-block cipher assembles plaintext content and a message authentication code (MAC)[8] into a *GenericBlockCipher*. As of TLS 1.1 this construct includes an explicit IV (initialization vector) field, into which random data is assigned. The resulting IV becomes the first encrypted block of the block-chain. The number of bytes in the IV is identical to the particular cipher algorithm's cipher-key length.

With TLS-DIVA, the encryption process proceeds as normal, however the resulting (encrypted) IV block is further enciphered with the DIVA keystream before being issued to the transport peer. For the receiving endpoint, once the original (encrypted) IV block is deciphered with the DIVA keystream, CBC deciphering proceeds as normal.

Successful CBC deciphering requires the correct IV block, otherwise the ciphersuite will report an error resulting in a fatal alert having a *decrypt_error* alert description.

Note that, if employing encrypt-then-MAC[8], the MAC is calculated *after* the IV is enciphered with DIVA so as to maintain the transparency of the DIVA encryption.

For a given TLSCiphertext record, keystream output proceeds from a location starting with the *ServerDIVA* (defined below) *stream_offset* field value plus 28 (the length of the ServerHello random_bytes sub-field). To this, each record advances the stream position by the IV length. Thus, once established, keystream position is known by both endpoints and need not be further conveyed over the transport.

Note that, since each endpoint maintains separate sequence numbers for client and server TLS records, each endpoint must maintain two independent keystreams.

For every TLS record so encrypted and decrypted, consumption of the client and server DIVA keystreams implicitly advances their internal positions. Although keystream position may be calculated as above, a DIVA keystream has an upper bound far exceeding that supported by a 64-bit number and is, therefore, managed internally.

Note that as the keystream position advances it may “wrap around”, however this scenario requires no special consideration on the part of the stream consumer.

C. Client and Server Hello DIVA Extensions

A new extension type, *wnlabs_diva* is defined and may be included by the client in its *ClientHello* message. Note the protocol-assigned value of this extension is, as of Spring 2017, yet to be formally assigned.

In order to provide DIVA support for the TLS session, clients may offer an extension of type wnlabs_diva in the (extended) ClientHello message. The *extension_data* field of this extension shall contain a structure, ClientDIVA, which encapsulates two fields: *serial number* and *diva_options*.

The client specifies the DIVA key to be used via the ClientDIVA serial_number field. This number must uniquely identify the particular DIVA key within the scope defined by the issuing authority, and must be known by both the client and the server in order for a successful TLS-DIVA session to be established.

A TLS protocol vector of at most 255 values, *Options* specifies at most 255 optional TLS-DIVA features. Options may include any of the following features (specified once each):

- The *server_offset* option (feature number one) indicates the client does not track, or does not require the server to track the keystream offset between TLS-DIVA sessions. Instead, the server chooses the session's initial keystream offset; in such cases it is recommended the server use a cryptographically-strong random number generator to compute the keystream offset. A typical TLS-DIVA session will employ this option.
- The *large_offset* option (feature number two) indicates support for a large, variable-length starting keystream offset rather than a fixed, 64-bit offset. If the client supports a large offset, it must include this option; if both the client and server support large offsets, the server should use a large offset.
- The *cipher_option* option (feature number three) indicates the ciphersuite negotiated for use by the TLS session will employ DIVA in the encryption and decryption process. Particulars of the method by which a keystream is employed by a ciphersuite are defined on a per-ciphersuite basis. For instance, chain-block ciphers mix the encrypted IV with keystream data before issuing the resultant GenericBlockCipher to the connection peer.

The client specifies supported options in the ClientDIVA diva_options field, or specifies an empty vector if the client does not support additional DIVA functionality. For instance, if the client's preferred ciphersuite does not support the wnlabs_diva extension, the client would not include cipher_option in this vector.

A *Number* is a vector of from 4 to 12 octets (32-96 bits) comprising a serial number uniquely associated with a DIVA key established by some trusted authority.

A server that receives an (extended) ClientHello message containing the wnlabs_diva extension, may use the information contained in the extension to employ DIVA in the

TLS session. In this event, the server shall include an extension of type wnlabs_diva in the (extended) ServerHello message. The extension_data field of this extension shall contain a structure, *ServerDIVA*, which encapsulates two fields: *stream_offset* and *diva_options*.

The ServerDIVA diva_options field specifies the optional DIVA features selected on the part of the server, which options must be honored by both client and server. If a TLS server is configured to require the enhanced security of TLS-DIVA, for any client not including the wnlabs_diva ClientDIVA extension as part of the (extended) ClientHello message, the server should respond with a fatal alert having an *access_denied* alert description.

If the client and server agree upon an empty set of options, the TLS handshake is modified for the purpose of peer authentication, but does not otherwise alter the TLS session.

Should the server require a DIVA feature that is not supported by the client, the server must respond with a fatal alert having an alert description of *handshake_failure*. Similarly, if the client is not willing to accept the server's chosen diva_options, the client must issue a fatal alert indicating a handshake failure.

The server provides the keystream offset in the stream_offset field. This offset indicates the initial position at which both the client and server will source DIVA keystream data for use in the modified TLS session handshake and for use also by any negotiated features.

There are two methods of specifying the stream_offset, distinguished by client-server associativity:

- In case of an implicit DIVA keystream offset, for instance if the DIVA key is employed exclusively by one client and one server, the server conveys its keystream offset for comparison on the part of the client.
- Alternatively, the DIVA keystream position is explicit, for instance if the DIVA key is employed by multiple, independent TLS-DIVA supporting servers. In this case the server selects a starting position intended to minimize or eliminate the risk of keystream offset collisions.

Note the absolute-first 64 keystream octets of a given DIVA key are reserved and *must not* be used to encipher any data other than that prescribed in the following. Accordingly, a stream_offset representing a position less than 64 is invalid, and must be rejected by the client.

The stream_offset is never sent in the clear, but rather is encoded according to the following procedure:

1. Obtain the 48 keystream octets generated at offset 10; these form one 16-octet value and one 32-octet value.
2. Provide, as input, the concatenation of the 16-octet value resulting from step one with the first 16 octets of the (enciphered) ServerHello *random* field and the 32 octets of the ClientHello *random* field to the SHA-256 hash function[9]; the output forms another 32-octet value.
3. Mix (exclusive-or) the octets of the original stream_offset value with the 32-octet value resulting from step two.
4. The 32-octet value obtained in step one is then used to encipher (exclusive-or) the result of step three, yielding the value to be transmitted to the client.

Encoding the transmitted stream_offset value in this fashion safeguards the privacy of the corresponding keystream. Furthermore, factoring both the ServerHello and ClientHello random data into the encoding constitutes an additional safeguard against inferring or tampering with the initial stream offset.

Offset fields are specified as one of two types: *FixedOffset* or *LargeOffset*; both are 32 octets in size.

Unless the client and server explicitly negotiate a large initial keystream offset, the ServerDIVA stream_offset field is interpreted as a FixedOffset. In this case, the server selects a 64-bit value and assigns it to the FixedOffset *offset* sub-field. The remaining 24 octets (comprising the FixedOffset *unused* sub-field) *must* be filled with random values.

Should the client offer and the server select the large_offset option, the ServerDIVA stream_offset field is interpreted as a LargeOffset. In this case the server defines the initial keystream offset as a value comprised of at least 1 and at most 31 octets, and places this value in the LargeOffset *offset* sub-field. Any padding needed to force the length of the LargeOffset to be exactly 32 bytes is placed in the LargeOffset *unused* sub-field; the unused octets *must* be filled with random values. The LargeOffset *unused_length* sub-field is assigned a value such that the least significant 5 bits specify the count of unused octets.

Note that a zero-length offset is not valid and must be rejected, in which case the recipient must respond with a fatal alert having a handshake_failure alert description.

D. TLS-DIVA Session Management

Support for the TLS-DIVA extension has implications for TLS session management. TLS provides a mechanism to support the re-use of a cached, previously negotiated set of session details, as well as a mechanism to negotiate a new set of security details for an already established TLS session. How TLS-DIVA details are managed as part of overall session handling is now defined in terms of TLS session resumption and renegotiation.

1) Session Caching and Resumption

When attempting to resume a TLS session, a client will include any TLS extensions normally offered as part of the (extended) ClientHello message. If the server opts to resume some cached session it will, in general, ignore these client extensions. However, this behavior is defined on a per-extension basis and, in the case of TLS-DIVA, extension particulars are considered private to each connection. Thus, DIVA session details are not recorded in the server's session cache.

In short, whether initiating or resuming a TLS session, a DIVA supporting TLS client should offer the ClientDIVA extension, in which case a DIVA supporting TLS server response MUST include the ServerDIVA extension according to the procedure described above.

2) Session Renegotiation

Current versions of the TLS protocol support session renegotiation. As part of this process, a new set of TLS-DIVA session particulars may be negotiated, in which case the client will specify both a DIVA key and a set of DIVA options via the ClientDIVA extension in the (extended) ClientHello, as outlined above. Note that new TLS-DIVA security parameters may be negotiated without limitation of any previously negotiated.

In response the server may compose and issue a new ServerDIVA extension to the client as part of the (extended) ServerHello.

Should the client, when initiating a renegotiation, prefer to maintain (any) active TLS-DIVA session parameters, it *must not* offer the ClientDIVA extension. Similarly, should the server prefer not to alter (any) active TLS-DIVA session parameters, it *must not* include the ServerDIVA extension in its (extended) ServerHello message. In this case, the client should continue with the active TLS-DIVA session parameters, however it may instead opt to send a handshake_failure fatal alert and close the connection.

Note that, by implication, once negotiated and activated TLS-DIVA security parameters must remain in effect for the remainder of the TLS session.

Whenever the server includes a wnlabs_diva extension as part of the (extended) ServerHello message, the new TLS-DIVA details become pending on both client and server. In such cases the server must modify the 28-octet random_bytes sub-field of the ServerHello message as described above. Otherwise, the server must not so modify the random_bytes, proceeding instead with a normal TLS handshake.

Any negotiated TLS-DIVA cipher details are activated in conjunction with overall TLS session parameter activation, that is, upon receipt of a ChangeCipherSpec handshake message.

ACKNOWLEDGMENT

The authors would like to thank Albert Meyburgh of Whitenoise Laboratories for providing technical feedback on portions of the specification, and would like to thank Layton Perrin of Sequor Systems for assistance in evaluating key aspects of the extension's design.

REFERENCES

[1] K. Moriarty, B. Kaliski, J. Jonsson, A. Rusch, "PKCS #1: RSA cryptography specifications version 2.2," Internet Engineering Task Force, Request for Comments: 8017, November 2016

[2] A. Takayasu, N. Kunihiro, "A tool kit for partial key exposure attacks on RSA," in Handschuh H., Eds. Topics in Cryptology – CT-RSA 2017. CT-RSA 2017. Lecture Notes in Computer Science, vol 10159. Springer, Cham, 2017

[3] "Advanced encryption standard (AES)," National Institute of Standards and Technology, Federal Information Processing Standards Publication 197, November 2001

[4] A. Bogdanov, D. Khovratovich, C. Rechberger, K. Leuven, "Biclique cryptanalysis of the full AES," in Lee D.H., Wang X., Eds. Advances in Cryptology – ASIACRYPT 2011. ASIACRYPT 2011. Lecture Notes in Computer Science, vol 7073. Springer, Berlin, Heidelberg, 2011

[5] I. Traore, M. Liu, "Evaluation of whitenoise cryptosystem." Information Security and Object Technology (ISOT) Group, University of Victoria, Department of Electrical and Computer Engineering, Technical Report No ECE03-3, February 2003, unpublished

[6] E. Chattopadhyay, D. Zuckerman, "Explicit two-source extractors and resilient functions," Proceedings of the forty-eighth annual ACM symposium on Theory of Computing, 2016, pp.670-683

[7] D. Eastlake, "Transport layer security (TLS) extensions: extension definitions," Internet Engineering Task Force, Request for Comments: 6066, January 2011

[8] T. Dierks, E. Rescorla, "The transport layer security (TLS) protocol version 1.2," Internet Engineering Task Force, Request for Comments: 5246, August 2008

[9] D. Eastlake, T. Hansen, "US secure hash algorithms (SHA and SHA-based HMAC and HKDF)," Internet Engineering Task Force, Request for Comments: 6234, May 2011

[10] P. Gutmann, "Encrypt-then-MAC for transport layer security (TLS) and datagram transport layer security (DTLS)," Internet Engineering Task Force, Request for Comments: 7366, September 2014

[11] "Secure hash standard (SHS)," National Institute of Standards and Technology, Federal Information Processing Standards Publication 180-4, March 2012