Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: September 3, 2018

H. Wang, Ed.
Y. Yang
X. Kang
Huawei Technology Pte. Ltd.
March 2, 2018

Using Identity as Raw Public Key in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)
draft-wang-tls-raw-public-key-with-ibc-00

Abstract

This document specifies the use of identity as a raw public key in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS). The protocol procedures are kept unchanged, but cipher suites are extended to support Identity-based signature (IBS). The example OID tables in the RFC 7250 [RFC7250] are expanded with OIDs specific to the IBC-based signature algorithms.

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 https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 3, 2018.

Table of Contents

1. Introduction

DISCLAIMER: This is a personal draft and has not yet seen significant security analysis.

Traditionally, TLS/DTLS client and server exchange public keys endorsed by PKIX [PKIX] certificates. It is considered complicate and may cause security weaknesses with the use of PKIX certificates [Defeating-SSL]. To simplify certificates exchange, using RAW public key in TLS/DTLS has been specified in RFC 7250. That is, instead of transmitting a full certificate in the TLS messages, only public keys are exchanged between client and server. However, an out-of-band mechanism for public key and identity binding is assumed.

Recently, 3GPP has adopted the EAP authentication framework for 5G and EAP-TLS is considered as one of candidate authentication methods for private networks, especially for networks with a large number of IOT devices. For IOT networks, TLS/DTLS with RAW public key is particularly attractive, but binding identities with public keys might be challenging. The cost to maintain a large table for

identity and public key mapping at server side incurrs additional maintenance cost. e.g. devices have to pre-register to the server.

To simplify the binding between the public key and the entity presenting the public key, a better way could be using Identity-Based Cryptography(IBC), such as ECCSI public key specified in RFC 6507, for authentication. Different from X.509 certificates and raw public keys, a public key in IBC takes the form of the entity's identity. This helps eliminate the necessity of binding between a public key and the entity presenting the public key.

The concept of IBC was first proposed by Adi Shamir in 1984. As a special class of public key cryptography, IBC uses a user's identity as public key, avoiding the hassle of public key certification in public key cryptosystems. IBC broadly includes IBE (Identity-based Encryption) and IBS (Identity-based Signature). For an IBC system to work, there exists a trusted third party, PKG (private key generator) responsible for issuing private keys to the users. In particular, the PKG has in possession a pair of Master Public Key and Master Secret Key; a private key is generated based on the user's identity by using the Master Secret key, while the Master Public key is used together with the user's identities for encryption (in case of IBE) and signature verification (in case of IBS).

A number of IBE and IBS algorithms have been standardized by different standardization bodies, such as IETF, IEEE, and ISO/IEC. For example, IETF has specified several RFCs such as RFC 5091 [RFC5091], RFC 6507 [RFC6507] and RFC6508 [RFC6508] for both IBE and IBS algorithms. ISO/JTC and IEEE also have a few standards on IBC algorithms.

RFC 7250 has specified the use of raw public key with TLS/DTLS protocol. Example OIDs for RSA, DSA, ECDSA algorithms have been given. However, supporting of IBS algorithms has not been included therein. Since IBC algorithms are efficient in public key transmission and also eliminate the binding between public keys and identities, in this document, an amendment to RFC 7250 is added for supporting IBS algorithms.

The document is orgranized as follows: Section 3 explains the use of identity as raw public key when IBS algorithms are chosen as the underlying digital signature mechanism, and example OIDs for IBS algorithms are given. Section 4 discusses provision of the global parameters used with the IBS algorithms. Section 5 discusses the security considerations.

2. Terms

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. Extension of RAW Public Key to IBC-based Identity

In RFC 7250, a new Certificate structure is defined with two types, X.509 and RawPublicKey. When raw public key is used in TLS, RawPublicKey type is selected and a data strucutre subjectPublicKeyInfo is used to specify the raw public key and its cryptographic algorithm. Within the subjectPublicKeyInfo structure, two fields, algorithm and parameters, are defined. The algorithm specifies the cryptographic algorithm used with raw public key, which is represented by an object Identifiers (OID); and the parameters field provides necessary parameters associated with the algorithm.

```
subjectPublicKeyInfo  ::=  SEQUENCE  {
        algorithm                       AlgorithmIdentifier,
        subjectPublicKey                BIT STRING }

   AlgorithmIdentifier   ::=  SEQUENCE  {
     algorithm               OBJECT IDENTIFIER,
     parameters              ANY DEFINED BY algorithm OPTIONAL  }
```

Figure 1: SubjectECCSIPublicKeyInfo ASN.1 Structure

When using an IBS algorithm, an identity is used as raw public key, which can be coverted to an OCTET string. Therefore, the Certificate and subjectPublicKey structure can be reused without changes. No OID for an IBS algorithm has been given as examples in [RFC 7250]. It is known that there are a few standardized IBS algorithms, therefore, in what follows several exmaples of OIDs for IBS algorithms are given.

The OIDs for some IBC-based Signature algorithmsare listed in the following table.

Key Type	Document	OID
ISO/IEC 14888-3 ibs-1	ISO/IEC 14888-3: IBS-1 mechansim (Identity-Based Signature)	1.0.14888.3.0.7
ISO/IEC 14888-3 ibs-2	ISO/IEC 14888-3: IBS-2 mechansim (Identity-Based Signature)	1.0.14888.3.0.8
SM9-1 Digital Signature Algorithm	SM9-1 Digital Signature Algorithm	1.2.156.10197.1.302.1
Elliptic Curve-Based Signatureless For Identitiy-based Encryption (ECCSI)	Section 5.2 in RFC 6507	1.3.6.1.5.x (need to apply)

Table 1: Algorithm Object Identifiers

In particular, ISO/IEC 14888-3 specifies two IBS algorithms, IBS-1 and IBS-2. The ECCSI is an IBS algorithm that is specified in IETF [RFC 6507]. SM9-1 is a Chinese standard for an IBS algorithm. Recently it has been accepted by ISO/IEC 14888-3

How are the paramters of AlgorithmIdentifier specified?

4. Parameters for Signature Verification

Using IBS algorithm in TLS/DTLS for raw public key exempts client and server from public key certification and identity binding. This is achieved by checking an entity’s signatures and its identity against the master public key of its PKG. With IBS algorithm, a PKG generates private keys for entities based on their identities. Global parameters such as PKG’s Master Public Key (MPK) need be provisioned to both client and server side. These parameters are not user specific, but PKG specific.

For a client, PKG specific parameters can be provioned, at the time PKG provisons the private key to the client. For the server, how to get the PKG specific parameters provisioned is out of the scope of this document, and it is depolyment dependent.

5. New Key Exchange Algorithms and Cipher Suites

To support identity as raw public key, new key exchange algorithms corresponding to the IBS algorithms need to be defined. The signing capability of the IBS algorithms is to be exploited, thus existing key exchange algorithms making use of ephemeral DH are extended to accomodate the support of the IBS algorithms. Considering the performance and the compatibility with the use of ECDSA in TLS (see RFC 4492), this specification proposes to support the IBS algorithm, ECCSI, defined in RFC 6507 [RFC6507]. As a reult, the table below summarizes the new key exchange algorithm, which mimics ECDHE_ECDSA (see RFC 4492).

Key Exchange Algorithm	Description
ECDHE_ECCSI	Ephemeral ECDH with ECCSI signatures

Table 2: Algorithm Object Identifiers

Note: The specification of ECDHE_ECCSI can follow ECHDE_ECDSA by substituting ECDSA with ECCSI. The detailed specification will be provided in the future

Note: Other key exchange algorithm with other IBS algorithm may be added in the future.

Accordingly, below defines the new cipher suites that use the above new key exchange algorithms.

CipherSuite TLS_ECDHE_ECCSI_WITH_AES_128_CBC_SHA256 = { 0xC0, 0x80 }

CipherSuite TLS_ECDHE_ECCSI_WITH_AES_256_CBC_SHA256 = { 0xC0, 0x8A }

6. TLS Client and Server Handshake Behavior

The handshake between the TLS client and server follows that defined in RFC 7250 [RFC7250], but with the support of the new key exchange algorithm and cipher suites due to the introducton of ECCSI. The high-level message exchange in the below figure shows TLS handshake using raw public keys, where the client_certificate_type and server_certificate_type extensions added to the client and server hello messages (see Section 4 of RFC 7250).

```
      client_hello,
      client_certificate_type,
      server_certificate_type   ->

                                <-  server_hello,
                                    client_certificate_type,
                                    server_certificate_type,
                                    certificate,
                                    server_key_exchange,
                                    certificate_request,
                                    server_hello_done
      certificate,
      client_key_exchange,
      certificate_verify,
      change_cipher_spec,
      finished                  ->

                                <- change_cipher_spec,
                                   finished

    Application Data        <------->     Application Data
```

Figure 2: Basic Raw Public Key TLS Exchange

6.1. Client Hello

If the TLS client wants to use ECCSI, then the client_certificate_type is set to be RawPublicKey. If the TLS client prefers accepting the server to use ECCSI, then the the server_certificate_type is set to be RawPublicKey, and the CipherSuite element of the client hello message is set to be the cipher suite(s) supporting ECCSI.

6.2. Server Hello

If the server receives a client hello that contains the client_certificate_type extension and/or the server_certificate_type extension, then three outcomes are possible [RFC 7250]:

1. The server does not support the extension defined in this document. In this case, the server returns the server hello without the extensions defined in this document.

2. The server supports the extension defined in this document, but it does not have any certificate type in common with the client. Then, the server terminates the session with a fatal alert of type "unsupported_certificate".

3. The server supports the extensions defined in this document and has at least one certificate type in common with the client. In this case, the processing rules described below are followed.

The client_certificate_type extension in the client hello indicates the certificate types the client is able to provide to the server, when requested using a certificate_request message. If the TLS server wants to request a certificate from the client (via the certificate_request message), it MUST include the client_certificate_type extension in the server hello. This client_certificate_type extension in the server hello then indicates the type of certificates the client is requested to provide in a subsequent certificate payload. The value conveyed in the client_certificate_type extension MUST be selected from one of the values provided in the client_certificate_type extension sent in the client hello. The server MUST also include a certificate_request payload in the server hello message.

If the server does not send a certificate_request payload or none of the certificates supported by the client match the server-supported certificate types, then the client_certificate_type payload in the server hello MUST be omitted.

If the server_certificate_type extension in the client hello is set be RawPublicKey and the CipherSuite element of the client hello message is set to be the cipher suite(s) supporting ECCSI, and the server chooses to use ECCSI, then the TLS server MUST place the SubjectPublicKeyInfo structure containing the ECCSI key into the Certificate payload. With the server_certificate_type extension in the server hello, the TLS server indicates the certificate type carried in the Certificate payload.

6.3. Client Authentication

When the TLS server has specified RawPublicKey as the client_certificate_type, and the TLS client sends the SubjectPublicKeyInfo structure containing an ECCSI key in the client certificate, authentication of the TLS client to the TLS server is achieved.

6.4. Server Authentication

When the TLS server has specified RawPublicKey as the server_certificate_type, and sends the SubjectPublicKeyInfo structure containing an ECCSI key in the server certificate, authentication of the TLS server to the TLS client is achieved.

7. Examples

In the following, examples of handshake exchages using ECCSI under RawPublicKey are illustrated.

7.1. TLS Client and Server Use ECCSI

In this example, both the TLS client and the TLS server use ECCSI, and they are restricted in that they can only process ECCSI keys. As a result, the TLS client sets the server_certificate_type extension to be raw public key while omits the client_certificate_type extension; in addition, the TLS client sets the ciphersuites in the client hellow messag to be TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA256, as shown in (1).

When the TLS server receives the client hello, it processes the message. Since it has an ECCSI key, it indicates in (2) that it agreed to use ECCSI and provided an ECCSI key by placing the SubjectPublicKeyInfo structure into the Certificate payload back to the client (3). The TLS server demands client authentication, and therefore includes a certificate_request (4). The client_certificate_type payload in (5) indicates that the TLS server accepts a raw public key. The TLS client, which has an ECCSI key, returns its ECCSI key in the Certificate payload (6) to the server.

```
client_hello,
cipher_suites=(TLS_ECDHE_ECCSI_WITH_AES_256_CBC_SHA256) // (1)
client_certificate_type=(RawPublicKey) // (1)
server_certificate_type=(RawPublicKey) // (1)
                         ->
                         <-  server_hello,
                             server_certificate_type=RawPublicKey // (2)
                             certificate, // (3)
                             client_certificate_type=RawPublicKey // (5)
                             certificate_request, // (4)
                             server_key_exchange,
                             server_hello_done

certificate, // (6)
client_key_exchange,
change_cipher_spec,
finished                   ->

                         <- change_cipher_spec,
                            finished

Application Data        <------->     Application Data
```

Figure 3: Basic Raw Public Key TLS Exchange

7.2. Combined Usage of Raw Public Keys and X.509 Certificates

This example combines the uses of an ECCSI key and an X.509 certificate. The TLS client uses an ECCSI key for client authentication, and the TLS server provides an X.509 certificate. This exchange starts with the client indicating its ability to process a raw public key, or an X.509 certificate, if provided by the server. It prefers a raw public key, since the RawPublicKey value precedes the other value in the server_certificate_type vector. Furthermore, the client indicates that it has a raw public key for client-side authentication (see (1)). The server chooses to provide its X.509 certificate in (3) and indicates that choice in (2). For client authentication, the server indicates in (4) that it has selected the raw public key format and requests a certificate from the client in (5). The TLS client provides an ECSSI key in (6) after receiving and processing the TLS server hello message.

```
client_hello,
cipher_suites=(TLS_ECDHE_ECSSI_WITH_AES_256_CBC_SHA256, TLS_ECDHE_ECDSA_WITH
_AES_256_CBC_SHA256) // (1)
server_certificate_type=(RawPublicKey, X.509) // (1)
client_certificate_type=(RawPublicKey) // (1)
                         ->
                         <-  server_hello,
                             server_certificate_type=X.509 // (2)
                             certificate, // (3)
                             client_certificate_type=RawPublicKey // (4)
                             certificate_request, // (5)
                             server_key_exchange,
                             server_hello_done
certificate, // (6)
client_key_exchange,
change_cipher_spec,
finished                  ->

                          <- change_cipher_spec,
                             finished

Application Data        <------->     Application Data
```

Figure 4: Basic Raw Public Key TLS Exchange

8. Security Considerations

Using IBS-enabled raw public key in TLS/DTLS will not change the handshake flows of TLS, so the security of the resulting protocol rests on the security of the used IBS algorithms. The example IBS algorithms mentioned above are all standardized and open, and thus the security of these algorithms is supposed to have gone through wide scrutinization.

9. IANA Considerations

This document describes new OIDs for IBS algorithms (Section 4), new key exchange algorithm (Section 5) and the corresponding new cipher suites (Section 5).

10. Acknowledgements

11. References

11.1. Normative References

[PKIX] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List(CRL) Profile", June 2008.

[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>.

[RFC5091] Boyen, X. and L. Martin, "Identity-Based Cryptography Standard (IBCS) #1: Supersingular Curve Implementations of the BF and BB1 Cryptosystems", RFC 5091, DOI 10.17487/RFC5091, December 2007, <https://www.rfc-editor.org/info/rfc5091>.

[RFC6507] Groves, M., "Elliptic Curve-Based Certificateless Signatures for Identity-Based Encryption (ECCSI)", RFC 6507, DOI 10.17487/RFC6507, February 2012, <https://www.rfc-editor.org/info/rfc6507>.

[RFC6508] Groves, M., "Sakai-Kasahara Key Encryption (SAKKE)", RFC 6508, DOI 10.17487/RFC6508, February 2012, <https://www.rfc-editor.org/info/rfc6508>.

[RFC7250] Wouters, P., Ed., Tschofenig, H., Ed., Gilmore, J., Weiler, S., and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7250, DOI 10.17487/RFC7250, June 2014, <https://www.rfc-editor.org/info/rfc7250>.

11.2. Informative References

[Defeating-SSL] Marlinspike, M.,, "New Tricks for Defeating SSL in Practice", Feb 2009, <http://www.blackhat.com/presentations/bh-dc-09/Marlinspike/BlackHat-DC-09-Marlinspike-Defeating-SSL.pdf>.

Appendix A. Examples

Authors' Addresses

Haiguang Wang (editor)
Huawei Technology Pte. Ltd.
20 Secience Park Road, #3-30/31
Singapore 117687
SG

Phone: +65 6825 4200
Email: wang.haiguang1@huawei.com

Yanjiang Yang
Huawei Technology Pte. Ltd.
20 Secience Park Road, #3-30/31
Singapore 117687
SG

Phone: +65 6825 4200
Email: yang.yanjiang@huawei.com

Xin Kang
Huawei Technology Pte. Ltd.
20 Secience Park Road, #3-30/31
Singapore 117687
SG

Phone: +65 6825 4200
Email: xin.kang@huawei.com