JSON Key Exchange
draft-hallambaker-json-key-exchange-03

Abstract

The JSON Key Exchange

This document is also available online at http://mathmesh.com/Documents/draft-hallambaker-json-key-exchange.html [1].

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 October 13, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.
# JSON Web Service Binding v1.0

## 1. Introduction

This document describes a lightweight key agreement mechanism using between 2 and four Diffe-Hellman or Elliptic Curve Diffie-Hallman keys. The mechanism may be used establish a shared session key with authentication of any or none of the initiator and the responder.

The approach described is similar to that adopted in the X3DH Key agreement [X3DH] used in Signal.
The objective of the key exchange is limited to establishing a shared secret between two mutually authenticated parties that cannot be derived by any other parties and cannot be reconstructed by either of the parties after the ephemeral contributions have been deleted.

The key exchange is intended for use as one component in a multi-layer security approach in which comprehensive security is provided through use of encryption and authentication at multiple layers in the protocol stack.

[[This figure is not viewable in this format. The figure is available at http://mathmesh.com/Documents/draft-hallambaker-json-key-exchange.html [2].]]

Multi-layer security

Specifically, this key exchange is intended for use at the presentation layer (e.g. authenticate and encrypt HTTP message bodies) to establish keys for authentication and optional encryption of messages in Web Service transactions.

Data Layer Encryption of cryptographic keys, protocol configuration profiles.

Presentation Layer Authentication of parties to a Web Service transaction.

Transport Layer Protect metadata against interception at single point on the message path (link or node)

Link Layer Protect messages against traffic analysis by means of interception at multiple points on the message path

2. Definitions

This section presents the related specifications and standard, the terms that are used as terms of art within the documents and the terms used as requirements language.

2.1. Requirements Language

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 [RFC2119].
2.2. Defined Terms

No terms of art are defined.

2.3. Related Specifications

JSON Key Exchange is used extensively in the Mathematical Mesh and related protocols [draft-hallambaker-mesh-architecture].

2.4. Implementation Status

The implementation status of the reference code base is described in the companion document [draft-hallambaker-mesh-developer].

3. Key Exchange Protocol

3.1. Parameters

The following parameters are defined

Algorithm: The key exchange protocol. Diffie Hellman in discrete
log, EdECDH25519 or EdECDH448.[RFC8032]

Key Derivation Function: The key derivation function. This is always
HKDF [RFC5869]

Key Wrap Function: The Key Wrap function. This is always RFC3394
[RFC3394]

It should be noted that the algorithm described makes use of the
Edwards form of the curve and not the Montgomery form described in
[RFC7748]. While these curves are isomorphic, implementations of the
Montgomery ladder do not lend themselves easily to the approach shown.

3.2. Notation

The notation adopted in [X3DH] is applied with minor modifications.

X || Y: The concatenation of the byte sequence X followed by the byte
sequence Y.

KE (PK1, PK2): The result of performing the key exchange (Diffie-
Hellman or Elliptic Curve Diffie Hellman) with the public
parameters of PK1 and the private parameters of PK2.

PK1+PK2: The public, private key pair formed by combining the public,
private parameters of PK1 and PK2.
Expand (PRK, info, L) The KDF expansion function that derives a key of length L bits from the Pre Random Key PRK using the string info as the salt.

Extract (IKM, salt) The KDF extraction function that derives a Pre-Random Key from the Initial Keying Material IKM and the salt value salt. If the salt value is specified as 0, the default salt of a string of zero bits the same length as the Pre-Random Key to be extracted is used.

3.3. Roles

Client The party that initiates the key exchange

Service The party that responds to the key exchange

3.4. Keys

IKC The identification key of the client. The encoding of this key MAY include one or more credentials such as a Mesh personal profile or an X.509 certificate binding the key to an identity.

IKS The identification key of the service. The encoding of this key MAY include one or more credentials such as a Mesh personal profile or an X.509 certificate binding the key to an identity.

EKC The ephemeral key of the client.

EKS The ephemeral key of the service.

3.5. Derived Keys

The key derivation function is used to derive separate keys for different purposes as shown below. The value L is the number of bits requires to key the algorithm specified. The salt value used to derive the PRK from the IKM is either the default salt value (all zeros) or the previous Rekey value as described below.

AK (PRK) Authentication Key = Expand (PRK, "authentication", L)

EK (PRK) Encryption Key = Expand (PRK, "encryption", L)

RA (PRK) Rekey Key = Expand (PRK, "rekey", L)

W (PRK) Witness value= Expand (PRK, "witness", L)
3.6. Initial Keying Request

A key exchange request is either an initial key exchange request or a rekeying request. An initial key exchange request MAY be issued at any time but a rekeying request cannot be sent until at least one initial keying request has been completed.

Rekey messages are authenticated under a Rekey shared secret established in a previous session. This may be the immediately preceding session or any prior session whose rekey token has not expired.

3.6.1. Initial Request Message

The client sends their identity and ephemeral key to the service. { IKC, EKC }

If the request message is an initial keying request, a credential associated with the identity key MAY be provided. The request MAY be authenticated by means of a digital signature.

3.6.2. Initial Response Message

The service calculates the IKM value as follows:

IKM = KE (IKC + EKC, IKS + EKS)

PRK = Extract (IKM, 0)

The service returns the values { IKS, EKS, W(IKM) } in a message authenticated under AK(IKM)

3.7. Rekeying.

Unless the key agreement is performed to a device of restricted capability, rekeying imposes minimal load on client or server and can be performed often, particularly if the Ed25519 curve is used for rekeying.

The state required for rekeying is separate from the keys used to encrypt and/or authenticate messages. This allows an application to store the rekeying key between communication sessions without risk of compromising the confidentiality or integrity of messages.

The use of the chaining salt ensures that rekeying cannot compromise the security of an already established key, even if a weaker key exchange algorithm is used. Thus a client MAY use an Ed488 key to
perform an initial key exchange and then perform rekey operations using a

3.7.1. Rekeying Request Message

The client sends a new ephemeral key to the service. { CEK }

The request message MUST be authenticated under the rekeying key of the shared secret of any unexpired session previously agreed. This allows the credentials of the parties to be omitted while providing the advantage of establishing a fresh forward secrecy session.

3.7.2. Rekeying Response Message

The service calculates the IKM value as follows:

IKM = KE (EKC, EKS)

The service returns the values { EKS, W(IKM) } in a message authenticated under AK(IKM)

PRK = Extract (IKM, RA’)

Where RA’ is the previous rekeying key output.

3.8. Initial Key Exchange Example

Alice requests access to a service using her account identifier alice@example.com. She has already registered her Mesh personal profile with the service where it is bound to her account identifier as the corresponding credential.

The Key exchange request is:
POST /.well-known/jwcexchange/HTTP/1.1  
Host: example.com  
Content-Length: 1068  

{
    "ExchangeRequest": {
        "ClientCredential": {
            "PublicKeyDH": {
                "kid": "MB50U-335BC-AUK02-E62ZE-4ME2J-IYUVJ",
                "Domain": "YE6bnq1M1X5ojaJto6PLP_PEwA",
                "Public": "5PxAR4YJvso7XS1m25OhhkSh_F5yopwHCC1xQZ04I2w5uc-twDyRhFPazUB BKkTG7ruS1qFJC-vuzcI6UL9Ee0QeJ9plnWoJA5CsoTFg_dHQVKEkdW5D227NT50ncE
AHlylnKSoIrRh4CXSG3MMC9oBOiJ7YF4oWSgj-T4ruMLXONac-o6T_2h-00d90P
Mpkj8_DcX3TdNwKnkSNJRrgJ5f8UnJU9C85Tr1eh-U7wzW753RaqwN3R-B3LVj1
t4d1qiKbgGEzSjknsrjtUIReuAYCI0fqOTap-6XcYDB_SHs4pZ8oEriU0aTB65
VTtje5fm16tp-308p7x6WQ"},
            "ClientNonce": {
                "PublicKeyDH": {
                    "kid": "MB626-GUF3J-RQRD3-IHIPS-UNOCI-HZDDR",
                    "Domain": "YE6bnq1M1X5ojaJto6PLP_PEwA",
                    "Public": "rVQmfx5-bS66pILwARcg_SyCB1NjZwW0yu9F7SE_2FuOJneQSqXo9bfz5UB2T
dD3W8wKjHJAPyvX05vhQcicGNKLB9MO5x4Pzn2K1wm0-W4jJNZ1qHjuy_181six
VqdG1RT4qg9LTG326BMmz2MV_bij_1z3qTa8vOb9WTXfK458OVOELphXmGSghi5t
x_rrS9z9gKR5a22qTrApHpdqJvIpX00HPGddn_sQ3K2DCTfV17aiwr7kwsO3p6
-NIHuAG4zRo86NE0UM_e6B4b7zX5cLt2n1_rqVNZwJOER1pRbxgIRtdJ34Hy8Vzv
uW1BMrZzT0JCR9g5kGG_OA"},
        "Domain": "YE6bnq1M1X5ojaJto6PLP_PEwA",
        "Public": "rVQmfx5-bS66pILwARcg_SyCB1NjZwW0yu9F7SE_2FuOJneQSqXo9bfz5UB2T
dD3W8wKjHJAPyvX05vhQcicGNKLB9MO5x4Pzn2K1wm0-W4jJNZ1qHjuy_181six
VqdG1RT4qg9LTG326BMmz2MV_bij_1z3qTa8vOb9WTXfK458OVOELphXmGSghi5t
x_rrS9z9gKR5a22qTrApHpdqJvIpX00HPGddn_sQ3K2DCTfV17aiwr7kwsO3p6
-NIHuAG4zRo86NE0UM_e6B4b7zX5cLt2n1_rqVNZwJOER1pRbxgIRtdJ34Hy8Vzv
uW1BMrZzT0JCR9g5kGG_OA"},
    "Domain": "YE6bnq1M1X5ojaJto6PLP_PEwA",
    "Public": "rVQmfx5-bS66pILwARcg_SyCB1NjZwW0yu9F7SE_2FuOJneQSqXo9bfz5UB2T
dD3W8wKjHJAPyvX05vhQcicGNKLB9MO5x4Pzn2K1wm0-W4jJNZ1qHjuy_181six
VqdG1RT4qg9LTG326BMmz2MV_bij_1z3qTa8vOb9WTXfK458OVOELphXmGSghi5t
x_rrS9z9gKR5a22qTrApHpdqJvIpX00HPGddn_sQ3K2DCTfV17aiwr7kwsO3p6
-NIHuAG4zRo86NE0UM_e6B4b7zX5cLt2n1_rqVNZwJOER1pRbxgIRtdJ34Hy8Vzv
uW1BMrZzT0JCR9g5kGG_OA"
}

Figure 1

The Keyu Exchange response is
HTTP/1.1 200 OK
Date: Wed 11 Apr 2018 09:01:08
Content-Length: 1360

{
  "ExchangeResponse": {
    "Status": 201,
    "StatusDescription": "Operation completed successfully",
    "Ticket": "
T3BhcXYVlI2hvbH1lIg91dHNpZGUgc2NvcGUgb2Ygc3B1Y21maWNhdGlvbQ",
  "Witness": "
tKE46P84C0qGur7cbZNlhxfvU7NSej98AKEIVh_2zzg",
  "ServerCredential": {
    "PublicKeyDH": {
      "kid": "MDD5A-RFNE7-JNUOS-23LTY-O4NIJ-PAYKM",
      "Domain": "
W-SCGVPsf1zgtgj.hw1ILLt_CQmYttxn4Ze4r18B5UO-a8G1XxeQEUjUvRdgDMX
6K1WbrctkJXa2pIaBMM3k9Ez1km1ivydaXn5s-bBEHJpuqWzax46IEt_4G9Wppb
_Qr0j4gFqrr0Fr7uIUoihF8byL7b4M69dvRT9we1KQSw0hDcnmWqN0x0ICWcY
v3D7GtM1mMiFkkgzv7csh-4UsMYDFqXq-DgGLzd00kvN5CFqS03G9D91DT9R4
98TQgLK8YDEj08_i2ADAGiP_GwHvDZHkqo0jFkgris5JJvEbfXg7S7h8yYEHzgUPL
7xKt1vGcY5s5ZQzHu8X6RgG"},
    "ServerNonce": {
      "PublicKeyDH": {
        "kid": "MBD52-J4WFI-XKDUI-5BG7-VHCQK-ZEHI7",
        "Domain": "
YE6bnq1MLX5ojaJto6PLP_PEwA",
      "Public": "
FkxMhJ64ZzdWl9QoBaeFenB-9BupOu-B0FLmGB9kkusumORQI-qiYQiGkEH93hH
qSe10LGuM83Vzc-SSU3QKm1Dbv1rbRSCqXbkLCnc8KHb0p0R8vinVI8x1cYLhp
_k8N5EjGmGcwozGzOxzK0xl7d4sXEXv_Djr5HfF57F41zAb34-ny2ZsD2jmGy4GR
526bMr69ceElwEpXx5_rC3CljJfc1AcAdbM5sZ31Fokurzq9X0F2ePeYReP93Ytbn
opzWmSyQ2koFg8Is9vnk6Dzy0mRHALpY0L6cmEEnyWvNExV7uGLMzsRtRLVDHvAK
q2Vulh-CxthTDlW4FWaA"},
      "Encryption": ["A256CBC-HS512"],
      "Authentication": ["HS512"]
    }
  }
}

Figure 2

Note that the example has the witness value but does not authenticate the signed result at present. Perhaps it would be better to create the witness value from the ticket data which eliminates the need for authenticating the response??
3.9. Rekey Example

4. Key Exchange Service

Supports key exchange to establish a shared secret and bound ticket between a client and a service

HTTP Well Known Service Prefix: /.well-known/jwcexchange

Every Recrypt Service transaction consists of exactly one request followed by exactly one response.

4.1. Shared classes

4.1.1. Structure: Algorithms

Describes an algorithm suite. Each suite consists of sets of authentication and encryption algorithms which are mutually compatible. i.e. the counterparty MAY select any one of the encryption algorithms and use it with any one of the authentication algorithms.

Encryption: String [0..Many] Algorithm identifiers of encryption and authenticated encryption algorithms offered

Authentication: String [0..Many] Authentication algorithm offer

4.2. Utility Transactions

4.3. Transaction: Exchange

Request: ExchangeRequest

Response: ExchangeResponse

Perform Key Exchange to establish shared key bound to a ticket.

4.3.1. Message: ExchangeRequest

Initiate the key exchange request.

Offer: Algorithms [0..Many] Set of message authentication and encryption algorithms offered by the client
4.3.2. Message: ExchangeResponse

Returns the server parameters.

Ticket: Binary (Optional)  Opaque session identifier.

Witness: Binary (Optional)  Opaque witness value used to prove binding to the ticket.

Encryption: String [0..Many]  Algorithm identifiers of encryption or authenticated encryption algorithm chosen

Authentication: String [0..Many]  Algorithm identifiers of authentication algorithm chosen

5. Security Considerations

6. IANA Considerations

The following registrations are required:

HTTP Content Coding Registry  jose-jwb

Well-Known URIs  /.well-known/srv/

[Or change registry to FCFS]

7. References

7.1. Normative References


7.2. Informative References

[draft-hallambaker-mesh-developer]


7.3. URIs


Author’s Address

Phillip Hallam-Baker
Comodo Group Inc.

Email: philliph@comodo.com