Network Working Group
Internet-Draft
Expires: August 28, 2008

M. Nakhjiri
Motorola
Y. Ohba
Toshiba

February 25, 2008

Derivation, delivery and management of EAP based keys for handover and re-authentication
draft-ietf-hokey-key-mgm-03

Status of this Memo

Copyright Notice

Abstract

This document describes a framework and a mechanism for deliverying usage specific root keys (USRK and DSUSRK), derived as part of an EAP EMSK hierarchy, and delivered from a server to an intended third party key holder. The framework description includes different scenarios for key delivery, depending on the type of keys being delivered. It also includes, specification of derivation of keys

required for security protection of key requests and delivery signaling. The mechanism description includes the definition for a three-party key distribution exchange (KDE) protocol.

Table of Contents

1. Introduction and Problem statement 3
2. Terminology 4
3. Key Delivery Architecture 5
3.1. Three Party Key Distribution Exchange (KDE) 7
3.2. Derivation of keys protecting the KDE 10
3.3. Specification of context and scope for distributed keys . 11
3.4. Automated key management for KIts and KCts 11
3.5. Key distribution exchange scenarios 12
4. KDE Message Format 13
4.1. Message Syntax 14
4.2. Message Encoding 17
5. Security Considerations 17
5.1. Security Association between Server and Third Party . . . 17
5.2. Replay Protection 18
5.3. Distribution of Duplicate Kpt 18
6. IANA consideration 18
7. Acknowledgements 19
8. References 19
8.1. Normative References 19
8.2. Informative references 20
Appendix A. KDE Transport 20
A.1. KDE Transport over UDP 20
A.2. KDE Transport over AAA 20
Authors' Addresses 21
Intellectual Property and Copyright Statements 22

1. Introduction and Problem statement

The ability of Extensible Authentication Protocol (EAP) framework [RFC3748] in incorporating desired authentication methods and generating master session keys (MSK and EMSK) [I-D.ietf-eap-keying] has led to the idea of using MSK and/or EMSK for bootstrapping further keys for a variety of security mechanisms. Especially, the MSK has been widely used for bootstrapping the wireless link security associations between the peer and the network attachment points. Issues arising from the use of MSK and the current bootstrapping methods when it comes to mobility performance and security are described in [I-D.ietf-hokey-reauth-ps]. Thus new efforts are under way to use EMSK instead of MSK for bootstrapping of keys for future use cases [I-D.ietf-hokey-emsk-hierarchy], [I-D.ietf-hokey-erx]. For instance [I-D.ietf-hokey-emsk-hierarchy] defines ways to create usage specific root keys (USRK) for bootstrapping security of a specific use case. [I-D.ietf-hokey-emsk-hierarchy] also defines ways to create domain specific root keys for bootstrapping security of a set of services within a domain.

Along with those lines, this document on the other hand provides a specification of a mechanism for secure delivery of such EMSK child keys from the EAP server, holding the EMSK, to the intended third party destinations. This is to address the following concerns:

1. EAP authentication is a 2 party protocol executed between an EAP peer and an EAP server and the EMSK is only generated and held at these two parties [I-D.ietf-eap-keying], while USRK and DSUSRK are also generated only by these two parties, but they typically need to be stored and utilized at third party key holders (e.g. AAA servers/entities) that are logically or even physically separate from the EAP server or peer. For instance, handover keying and re-authentication service requires distribution of keys a variety of intermediaries. This would mean these root keys need to be delivered to these third party key holders (KH) in a secure manner, while considering the requirements stated in [RFC4962]

2. EAP authentication and EMSK generation process is oblivious to the service and authorization requests following the initial EAP authentication. Thus at the time of EAP authentication, the EAP parties do not have access to the input data required for creation of the USRK or DSUSRK [I-D.ietf-hokey-emsk-hierarchy]. Such input data is typically acquired and delivered to a server (the EAP server or DSR-KH) at a later stage. The server then performs the derivation function, followed by a secure delivery of the resulting keys to these third party key holders.

One purpose of this document is to show how the required input data for root key derivation can be delivered to the server, and how the generated key material is delivered to the third party key holder in a secure manner. The specification also includes derivation of key material required for secure delivery and channel binding procedures for these key materials to ensure that not only the keys are not exposed to unintended parties during delivery, but also the scope and usage context for the key is properly understood and agreed upon by the initial parties.

Another purpose of this document is to provide exact syntax for a three-party key distribution exchange (KDE) protocol, a protocol used for delivering USRK and and DSUSRK from a server to a third-party.

2. Terminology

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].

USRK: Usage Specific Root key. A root key generated from EMSK and is used for a specific usage that is authorized for a peer, following an EAP authentication. USRK is domain independent.

USR-KH: USRK holder. The USR-KH is responsible for receiving, holding and protection of the USRK derived directly from EMSK.

DSRK: Domain Specific Root key. A root key generated from EMSK and is used within a specific domain that EAP-authenticated peer is authorized to receive services from or roam into. DSRK is usage independent.

DSR-KH: DSRK holder. The DSRK holder is responsible for receiving, holding and protection of the DSRK derived directly from EMSK.

DSUSRK: Domain Specific Usage Specific Root key. A root key generated from DSRK and is used for a specific usage within a specific domain that an EAP-authenticated peer is authorized to receive services from. DSUSRK is both usage and domain dependent.

DSUSR-KH: DSRK holder. The DSUSRK holder is responsible for receiving, holding and protection of the DSUSRK.

IK and CK: Integrity and cipher keys, used to protect the key delivery signaling between the peer and the EAP server. These two keys are some times referred to as key delivery keys.

3. Key Delivery Architecture

The EAP server is only responsible for performing EAP authentication and is not expected to be involved in any service authorization decisions, neither is the EAP server aware of the future service requests at the time of authentication. The authorization decisions based on the user service profile and provisioning of services including support for service security is expected to happen by third parties, such as AAA servers or service servers. When EAP-based keying is used, such servers will cache and use the USRKs, DSRKs or DSUSRKs, generated from EMSK, as root keys for derivation of further keys to secure the services they are providing. Thus they are considered third party key holders (KH) with respect to the initial two EAP parties (EAP peer and server). However, since EMSK cannot be exported from EAP server, such third parties need to request the EAP server to generate the relevant root key (USRK) from the EMSK and deliver the requested key to them. The third party needs to provide the required input data to be used along with the pseudo random function (PRF) to the EAP server to generate the requested key. The following types of top level key holders can be envisioned:

USRK holder (USR-KH): An entity acting as a recipient and then holder of the usage specific root key (USRK). The USR-KH is possibly responsible for derivation and distribution of any child keys derived from USRK for that specific usage. It is possible that this USR-KH server is not physically disjunct from the EAP server but is simply considered as a separate logic to off-load the EAP server from the need to handle usage specific services, such as HOKEY service. However, to keep the security specifications generic here, we assume that USR-KH and EAP server are physically separate and specify the delivery of USRK from EAP server to USR-KH accordingly.

DSRK holder (DSR-KH): An entity acting as a recipient and then holder of the domain specific root key (DSRK). The DSR-KH is responsible for derivation and distribution of any child keys derived from DSRK for that specific domain. The most likely realization of DSR-KH is a AAA server in the corresponding domain, responsible for setting the policies for usage of DSRK within the domain.

DSUSRK holder (DSUSR-KH): An entity acting as a recipient and then holder of the domain specific and usage specific root key (DSUSRK) delivered from the EAP server. The DSUSR-KH is possibly responsible for derivation and distribution of any child keys derived from DSUSRK for that specific domain and usage. The most likely realization of DSUSR-KH is a AAA server in the corresponding domain, responsible for the service offered within

the domain for the specific usage at hand.

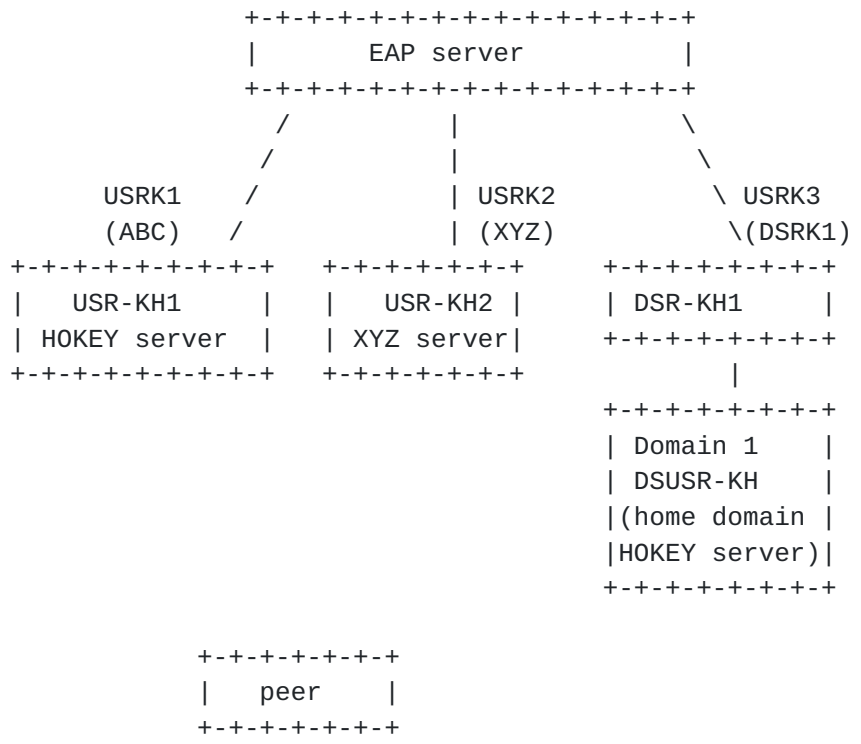

Figure 1: Key delivery for various EMSK child key cateories

As one can see, depending on the type of key being delivered, different third party key holders are involved. Each of the top level key holders (USR-KH, DSR-KH has a interface with the EAP server, for delivering usage specific (and/or domain specific) input data needed for root key generation (USRK, DSRK) to the EAP server and receiving the resulting root key from the EAP server. The DSUSR-KH is considered a second-level key holder and has an interface with DSR-KH.

Regardless of the type of key being delivered, the model for EAP based key derivation and delivery interface can be generalized as a 3 party key distribution model, since EAP authentication method signaling and the following EMSK generation is performed between the peer and the EAP server in a manner that is almost transparent to all intermediaries, while the EMSK is used to derive the top level root keys and deliver those to a third party key holder, such as USR-KH or DSR-KH.

3.1. Three Party Key Distribution Exchange (KDE)

In the following we describe the generic mechanism for a three- party key distribution exchange (KDE), where a key is distributed from a network server (with parent key holder) to a third party. The following shows a generic trust model for the key distribution mechanism to the third party. The peer (P) and a parent key holder, called "server" (S) in this model share a parent key (Kps) and a set of security associations (SA1) for integrity and privacy protection of signaling between the peer and the server (KIps and KCps). The goal of the keying solution is to use the parent key (Kps) and generate a child key (Kpt) to be shared between the peer and the third party intermediary (T). The peer is able to generate Kpt, but Kpt needs to be distributed to a third party intermediary (T). The goal of this section is to provide a the general description of the KDE (key distribution exchange) for distribution of Kpt from S to T. We also assume that the server (S) and the third party (T) share a similar set security association, SA2 (KIts, KCts).

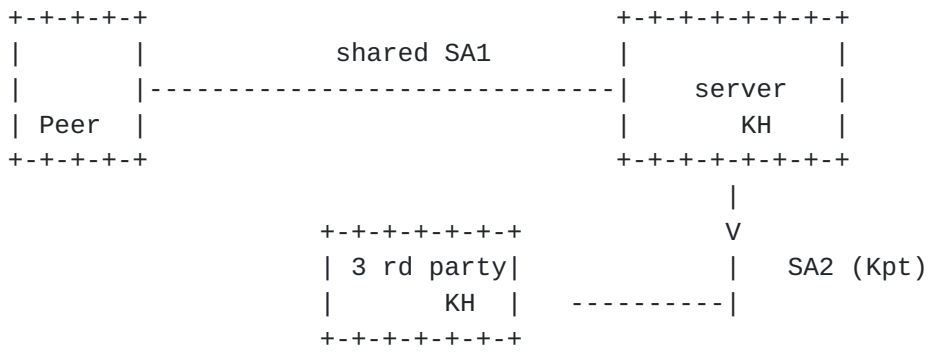

Figure 2: Distribution of a child key from a parent key key holder to a 3rd party child key key holder

The key distribution exchange described here meets the requirements for such 3-party lay-out, providing channel binding and avoids the lying intermediary scenario. The exchange proposed below is to perform a channel binding and avoid the lying intermediary scenario. The description below can be carried over a generic transport and thus is independent of the exact type of protocol that is used.

The key distribution to a third-party basically consists of 1 exchange, i.e. 2 messages between the peer and the server. However, in most scenarios each message traverses through the intermediary, i.e. Over two logical hops (peer-third party) and (third party-server) even though the exchange seems to consist of 4 logical messages. It should be noted that the information in message 0 is typically conveyed as an advertisement through other means and hence message 0 is optional.

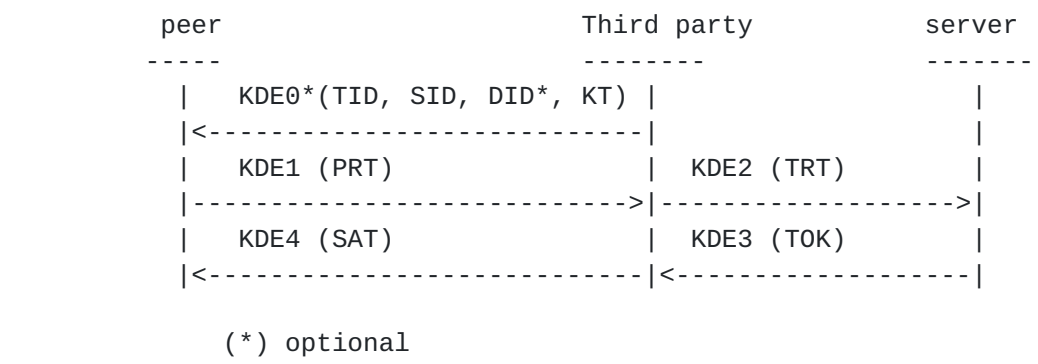

Figure 3: Handover using EAP-HR

KDE message 0 (optional): Third party sends its own identifier (TID), the Server ID (SID), the Domain ID (DID) and the KT (Key Type) to peer. These identifiers need to be recognizable by the server S and when AAA signaling is used may be carried as AAA attributes. KT is used for uniquely identifying the type of the key. DID is used to create domain specific keys or to assist in key distribution. DID is not included when the KT is other than 0 (DSRK) or 2 (DS-rRK).

KDE message 1: Peer sends a peer request token (PRT) to the third party including the TID reported by the third party and a freshness value. The contents of PRT are detailed below.

KDE message 2: Third party uses the PRT and creates a third party request token (TRT) and sends it to the server. The contents of TRT are detailed below.

KDE message 3: Server sends the Kpt to third party wrapped inside a token called Key Token (TOK). The TOK carries a Server Authorization Token (SAT) destined for the peer, carrying assurance for the peer that the server has sent the key Kpt to the properly identified third party (identified by TID).

KDE message 4: The third party extracts the SAT from the server and forwards it to the peer. The successful receipt of message 4 by peer means that the third party has successfully verified integrity of message 3 and decrypted Kpt.

{X}K: X encrypted with key K

Int[K, X]: X || MIC (K, X), where MIC Message Integrity Code over X with key K.

PRT : Int[KIps, (SEQps, PID, TID, SID, DID*, KT, KN_KIps)]

PRT (Peer Request Token) carries a sequence number (SEQps), the identities of the peer (PID), the server (SID), the third party (TID) and the domain (DID), the KT (Key Type) and the name of KIps along with the signature. The signature is called the peer request authenticator (PRA). KIps is a symmetric key shared between peer and Server for signing and identified by KN_KIps. SEQps is a sequence number generated by the peer and maintained between the peer and server. The initial sequence number starts from one (1). When the sequence number wraps up, then SA1 MUST be deleted and any KDE message MUST NOT be generated or processed thereafter. DID is not included when KT is other than 0 (DSRK) or or 2 (DS-rRK).

TRT : Int[KIts, (Nt, PID, TID, PRT)]

TRT (Third party Request Token) carries the token from the peer along with the third party and peer IDs and a signature for integrity protection. KIts is the shared key used for signing purposes. Nt is a nonce generated by the third party. Providing third party identifier both explicitly by the third party and both implicitly through PRT allows the server to detect a lying third party.

TOK : Int[KIts, (Nt+1, PID, TID, SID, DID*, KT, {Kpt, KN_Kpt, KL_Kpt}KCts, SAT)]

TOK(Key Token) carries the key to be distributed to the third party (Kpt) wrapped with an encryption key (KCts). KL_Kx is the key lifetime for key Kx.

SAT : Int[KIps,(SEQps+1, PID, TID, SID, DID*, KN_Kpt, KL_Kpt, KN_KIps)]

SAT (Server Authorization Token) carries assurance (in form of signature on the incremented nonce value) for the peer that the server has sent the key Kpt to the properly identified third party (identified by TID). DID is not included when KT is other than 0 (DSRK) or 2 (DS-rRK).

The exchange proposed above can avoid the lying intermediary scenario, as follows: if an intermediary decided to announce two different identifiers to the peer versus to the server, e.g. a down link ID to the peer (DTID) and a different uplink ID to the server (UTID). The peer uses DTID in its token towards the server, while the intermediary uses its UTID in its token to the server. Server must use the UTID from PRT to calculate the MIC in TRT and if there

is a match, then the server can verify that DTID and UTID are the same as the TID and proceed with generating and provisioning of Kpt, otherwise the server MUST return a failure code instead of generating an Kpt.

3.2. Derivation of keys protecting the KDE

As shown in the generic description of the key distribution exchange, to protect the exchange, at least one (or two) keys are required to protect the exchange. These keys are an integrity and a cipher key. These keys are generated from the EMSK hierarchy themselves. However, as discussed when enumerating the various KDE use case scenarios, the KDE can and need to be used in many different scenarios for delivering keys. Depending on the key that is being delivered, the integrity and cipher keys can be generated at different levels of the key hierarchy as well. For instance to protect the KDE performed to deliver a USRK, these two keys are generated directly from EMSK.

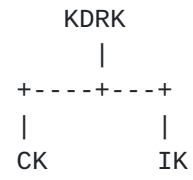

Figure 4: Key delivery keys as EMSK Child keys

Cipher key (CK) and Integrity Key (IK) are used to protect KDE for delivery of USRKs and DSUSRKs. CK and IK are generated from KDRK (Key Distribution Root Key) which is either EMSK or DSRK in the use cases described in this document. When KDRK is EMSK, CK and IK are defined as USRKs. When KDRK is DSRK, CK and IK are defined as DSUSRKs. The lengthes of CK and IK depends on actual integrity and encryption algorithms in use.

If KDRK is the EMSK, then CK and IK are defined using the USRK derivation algorithm defined in [I-D.ietf-hokey-emsk-hierarchy] as follows:

IK = USRK(usage="kde-integrity-key@ietf.org", length)

CK = USRK(usage="kde-cipher-key@ietf.org", length)

USRK(usage, length) is the USRK key derivation function with the usage and key length specified in the first and second parameters, respectively.

If KDRK is the DSRK for domain="example.net", then CK and IK are

defined using the DSUSRK derivation algorithm defined in [I-D.ietf-hokey-emsk-hierarchy] as follows:

IK = DSUSRK(usage="kde-integrity-key@ietf.org", domain, length)

CK = DSUSRK(usage="kde-cipher-key@ietf.org", domain, length)

DSUSRK(usage, domain, length) is the DSUSRK key derivation function with the usage, domain and key length specified in the first, second and third parameters, respectively.

If KDRK is other than the EMSK or DSRK, then CK and IK are defined using the KDF defined in [I-D.ietf-hokey-emsk-hierarchy] as follows:

IK = KDF(KDRK, "kde-integrity-key@ietf.org" + length)

CK = KDF(KDRK, "kde-cipher-key@ietf.org" + length)

In this case, the IK and CK names are defined as SHA-1-64(KDRK-name + "kde-integrity-key@ietf.org") and SHA-1-64(KDRK-name + "kde-cipher@ietf.org"), respectively, where SHA-1-64 is the first 64-octet of SHA-1.

3.3. Specification of context and scope for distributed keys

The key lifetime of each distributed key MUST NOT be greater than that of its parent key.

The key context of each distributed key is determined by the sequence of KTs in the key hierarchy. When a DSRK is being delivered and the DSRK applies to only a specific set of services, the service types may need to be carried as part of context for the key. Carrying such a specific set of services are outside the scope of this document.

The key scope of each distributed key is determined by the sequence of (PID, SID, TID, DID, KT)-tuples in the key hierarchy.

3.4. Automated key management for KIts and KCts

KIts and KCts require automated key management [RFC4107] since these are long-term session keys used by more than two parties. Kerberos [RFC4120] MAY be used as an automated key management protocol for distributing KIts and KCts. If there is no direct trust relationship between the third-party and the server, then inter-realm Kerberos MAY be used to create a direct trust relationship between the third-party and the server from a chain of trust relationships.

Note that the automated key management mechanism described above is

not required if hop-by-hop security is used for protecting KDE messages. See Section 5 for more discussion.

3.5. Key distribution exchange scenarios

As mentioned earlier, EMSK can be used to generate any of the USRKs, DSRKs and DSUSRKs. The following scenarios can be envisioned for distribution of a key to a 3rd party. All scenarios assume the peer and the EAP server have mutually authenticated to each other using an EAP method and have generated an EMSK. Since the EAP server performing EAP method authentication and EMSK generation resides in peer's home domain, for practical purposes, for the mechanisms described in this document, the USR-KH MUST reside in this domain. Note that other key distribution scenarios may also be possible since the key distribution protocol is designed to be generic.

Scenario 1: EAP server to USR-KH: In this scenario, an EAP server delivers a USRK to a USR-KH. The trigger and mechanism for key delivery may involve a specific request from the peer and another intermediary (such as authenticator). In the case of HOKEY re-authentication service, a DSRK or an rRK is distributed.

Scenario 2: DSR-KH to DSUSR-KH: In this scenario, a DSR-KH in a specific domain delivers keying material to the DSUSR-KH in the same domain. In the case of HOKEY re-authentication service, a DS-rRK is distributed.

The mapping between the protocol parameters in each scenario to the protocol parameters of the KDE protocol defined in Section 3.1 is given below, where IK_X and CK_X are IK and CK derived from key X, respectively. The keyName-NAI is defined in [I-D.ietf-hokey-erx].

KDE Param.	Scenario 1	Scenario 2
PID	keyName-NAI	
SID	EAP Server ID	DSR-KH ID
TID	USR-KH ID	DSUSR-KH ID
Kpt	USRK	DSUSRK
KIps	IK_EMSK	IK_DSRK
KCps	CK_EMSK	CK_DSRK
KIts	Any pre-existing key	
KCts	Any pre-existing key	

The key distribution exchanges for some of the above scenarios can be recursively combined into a single 1.5-roundtrip exchange. For example, a combined key distribution exchange for Scenarios 1 and 2 is illustrated in Figure 5 where KDE[1-4] and KDE'[1-4] are messages for Scenarios 1 and 2, respectively.

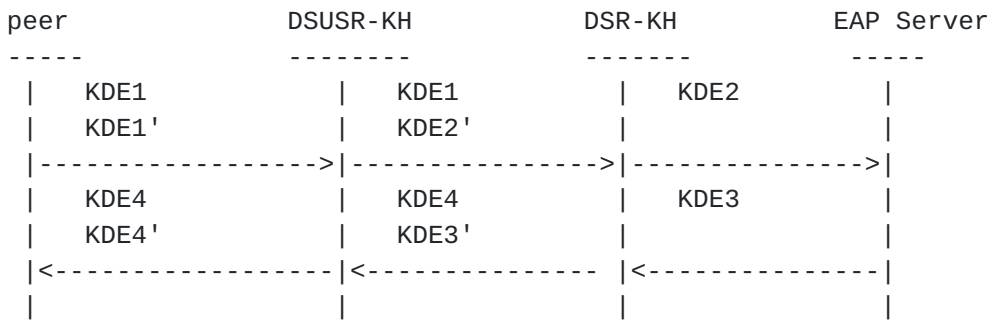

Figure 5: Combined Message Exchange

4. KDE Message Format

The format of KDE messages is defined here in terms of Abstract Syntax Notation One (ASN.1) [X680], which provides a syntax for specifying both the abstract layout of protocol messages as well as their encodings.

4.1. Message Syntax

The syntax of KDE messages is defined here in terms of Abstract Syntax Notation One (ASN.1) [X680], which provides a syntax for specifying both the abstract layout of protocol messages as well as their encodings.

```
-- OID for KDE
HokeyKdeV1 {
        iso(1) identified-organization(3) dod(6) internet(1)
        security(5) mechanisms(5) hokey(to be assigned by IANA)
        kde-v1(1)
} DEFINITIONS AUTOMATIC TAGS ::= BEGIN

-- OID arc for HOKEY KDE
--
-- This OID may be used to identify HOKEY KDE messages
-- encapsulated in other protocols.
--
-- This OID also designates the OID arc for HOKEY KDE-related OIDs.
--
id-hokey-kde-v1    OBJECT IDENTIFIER ::= {
        iso(1) identified-organization(3) dod(6) internet(1)
        security(5) mechanisms(5) hokey(to be assigned by IANA)
        kde-v1(1)
}

UInt32          ::= INTEGER (0..4294967295)
                    -- unsigned 32 bit values

KDE-PDU ::= SEQUENCE {
  version   INTEGER (1..255)
              -- Version of KDE protocol (=1 in this document)
  payload   KDE-Payload
              -- Payload of KDE message
}

-- A payload contains one or more KDE messages.

-- The payload contains one or more KDE messages.  Two or more KDE
-- messsage are allowed to support combined exchange.
KDE-Payload ::= SEQUENCE OF {
  CHOICE {
    kde0 KDE0  -- KDE0
    kde1 KDE1,  -- KDE1
    kde2 KDE2,  -- KDE2
    kde3 KDE3,  -- KDE3
    kde4 KDE4   -- KDE4
```

```
  }
}

KDE0 ::= SEQUENCE {
  tid    OCTET STRING,          -- Third-party ID
  sid    OCTET STRING,          -- Server ID
  did    OCTET STRING OPTIONAL, -- Domain ID
  keytype        INTEGER(0..255)
                     -- Key Type of Kpt (IANA assigned value)
}

KDE1 ::= SEQUENCE {
  prt PRT -- Peer Request Token
}

KDE2 ::= SEQUENCE {
  trt TRT -- Third Party Request Token
}

KDE3 ::= SEQUENCE {
  tot TOT -- Key Token
}

KDE4 ::= SEQUENCE {
  tot SAT -- Server Authorization Token
}

-- PRT (Peer Request Token)
PRT ::= SEQUENCE {
  seq            Uint32,       -- Sequence Number (intial value is 1)
  pid            OCTET STRING, -- Peer ID
  tid            OCTET STRING, -- Third-party ID
  sid            OCTET STRING, -- Server ID
  did            OCTET STRING OPTIONAL,
                               -- Domain ID
  keytype        INTEGER(0..255)
                     -- Key Type of Kpt (IANA assigned value)
  kips-name      OCTET STRING, -- Key name of KIps
  integrity-data IntegrityData
                     -- Integrity protection with KIps
}

-- TRT (Third party Request Token)
TRT ::= SEQUENCE {
  tnonce        Uint32        -- Third-party Nonce
  pid           OCTET STRING, -- Peer ID
  tid           OCTET STRING, -- Third-party ID
  prt    PRT
```

```
}

-- TOK (Key Token)
TOK ::= SEQUENCE {
  tnonce Uint32               -- Third-party Nonce+1
  pid            OCTET STRING, -- Peer ID
  tid            OCTET STRING, -- Third-party ID
  sid            OCTET STRING, -- Server ID
  did            OCTET STRING OPTIONAL,
                               -- Domain ID
  keytype        INTEGER(0..255)
                               -- Key Type of Kpt (IANA assigned value)
  enc-kpt        EnctyptedData
                   -- Kpt and its name and lifetime encrypted with KCts
  sat            SAT,
  integrity-data IntegrityData
                   -- Integrity protection with KIts
}

-- SAT (Server Authorization Token)
SAT ::= SEQUENCE {
  seq            Uint32,       -- Sequence Number + 1
  pid            OCTET STRING, -- Peer ID
  tid            OCTET STRING, -- Third-party ID
  sid            OCTET STRING, -- Server ID
  did            OCTET STRING OPTIONAL,
                               -- Domain ID
  kpt-name       OCTET STRING, -- Key name of Kpt
  kpt-lifetime   Uint32 -- Lifetime of Kpt in seconds
  kps-name       OCTET STRING, -- Key name of Kps
  integrity-data IntegrityData
                   -- Integrity protection with KIps
}

-- Integrity Data
IntegrityData ::= SEQUENCE {
  integrity-algorithm IntegirityAlgorithm, -- integrity algorithm
  mac                 OCTET_STRING -- message authentication code
}

-- Encrypted Data
EncryptedData ::= SEQUENCE {
  encryption-algorithm EncryptionAlgorithm, -- encryption algorithm
  cipher               OCTET_STRING         -- encrypted data
}

-- Encryption algorithm.  The data contains an IKEv2 Transform ID of
-- Transform Type 1 [RFC4306] for the encryption algorithm.  All KDE
```

```
 -- implementations MUST support ENCR_AES_CBC [RFC3602].  It is allowed
 -- to use null encryption (ENCR_NULL) for KDE2 and KDE3 in the case
 -- where hop-by-hop security between third party and server is
 -- acceptable.
 EncryptionAlgorithm ::= UInt32

 -- Integrith algorithm.  The data contains an IKEv2 Transform ID of
 -- Transform Type 3 [RFC4306] for the integrity algorithm.  All KDE
 -- implementations MUST support AUTH_HMAC_SHA1_160 (7) [RFC4595].
 -- It is allowed to use a null integrity mechanism (NONE) for
 -- for KDE2 and KDE3 in the case where hop-by-hop security between
 -- third party and server is acceptable.

 IntegrityAlgorithm ::= UInt32

 END
```

4.2. Message Encoding

The default encoding of KDE protocol messages shall obey the PER (Packed Encoding Rules) of ASN.1 as described in [X691]. A KDE transport protocol may specify other ASN.1 encoding method.

5. Security Considerations

5.1. Security Association between Server and Third Party

The key distribution mechanism described in this document is designed to work with both end-to-end and hop-by-hop security association between a server and a third party.

When end-to-end security is used, the key distribution mechanism assumes existence of a direct trust relationship between the server and the third party key holder. When such a direct trust relationship may be dynamically created from a chain of transitive trust relationships with the use of inter-realm Kerberos to distribute KIts and KCts as described in Section 3.4. Therefore, the key distribution method described in this document eliminates the need for hop-by-hop security associations along the transitive trust relationship.

When hop-by-hop security is used, no integrity protection and encryption is provided within the KDE protocol. A null encryption algorithm (ENCR_NULL) and a null integrity protection (NONE) are specified in KDE. In this case, underlying transport protocol security such as IPsec and (D)TLS MUST be used instead. The use of

hop-by-hop security implies that an intermediary on each hop can access the distributed key material. Hence the use of hop-by-hop security SHOULD be limited to an environment where an intermediary is trusted not to use the distributed key material.

5.2. Replay Protection

The KDE protocol defines two freshness values to provide replay protection. Sequence numbers generated by peer and maintained by peer and server provide anti-replay for KDE messages 1, 2 and 4. Nonces generated by third-party provide anti-replay for KDE message 3.

5.3. Distribution of Duplicate Kpt

If a Kpt is a USRK or a DSUSRK, it should be sufficient that distribution of the Kpt happens only once during the lifetime of it's root key, i.e., EMSK. Nevertheless, a peer may attempt to execute the KDE protocol multiple times via the same third party with specifying the same parameters in KDE message 1 except for sequence number, for some reason such as loss of key state due to temporal disconnection from the network. The server may accept such an attempt and distribute a copy of the same Kpt back to the third party, given that the lifetime of the Kpt (KL_Kpt) is recomputed such that the key expiration time of the Kpt will not change for all copies of the same Kpt.

6. IANA consideration

This document defines a new SMI Security for Mechanism Code for hokey(to be assigned by IANA).

This document defines a new SMI Security for Mechanism hokey Code for kde-v1(1).

This document defines new usage labels, such as those used in generation of CK and IK. The corresponding labels, i.e., "kde-cipher-key@ietf.org" for CK and "kde-integrity-key@ietf.org" for IK, need to be assigned numerical values by IANA.

The Key Type namespace is used to identify the type of Kpt. The range of values 0 - 65,535 are for permanent, standard message types, allocated by IETF Consensus [IANA]. This document defines the values 0 (DSRK), 1 (rRK) and 2 (DS-rRK).

7. Acknowledgements

The author would like to thank Dan Harkins, Chunqiang Li, Rafael Marin Lopez and Charles Clancy for their valuable contributions to the formation of the KDE.

8. References

8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, "Extensible Authentication Protocol (EAP)", RFC 3748, June 2004.

[I-D.ietf-hokey-emsk-hierarchy] Salowey, J., Dondeti, L., Narayanan, V., and M. Nakhjiri, "Specification for the Derivation of Root Keys from an Extended Master Session Key (EMSK)", draft-ietf-hokey-emsk-hierarchy-04 (work in progress), February 2008.

[I-D.ietf-hokey-erx] Narayanan, V. and L. Dondeti, "EAP Extensions for EAP Re-authentication Protocol (ERP)", draft-ietf-hokey-erx-12 (work in progress), February 2008.

[RFC3579] Aboba, B. and P. Calhoun, "RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP)", RFC 3579, September 2003.

[RFC4072] Eronen, P., Hiller, T., and G. Zorn, "Diameter Extensible Authentication Protocol (EAP) Application", RFC 4072, August 2005.

[RFC4120] Neuman, C., Yu, T., Hartman, S., and K. Raeburn, "The Kerberos Network Authentication Service (V5)", RFC 4120, July 2005.

[RFC4962] Housley, R. and B. Aboba, "Guidance for Authentication, Authorization, and Accounting (AAA) Key Management", BCP 132, RFC 4962, July 2007.

[X680] "Abstract Syntax Notation One (ASN.1): Specification of Basic Notation, ITU-T Recommendation X.680 (2002).",

July 2002.

[X691] "Abstract Syntax Notation One (ASN.1): ASN.1 encoding rules: Specification of Packed Encoding Rules (PER), ITU-T Recommendation X.691 (2002).", July 2002.

[IANA] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 2434, October 1998.

8.2. Informative references

[RFC4107] Bellovin, S. and R. Housley, "Guidelines for Cryptographic Key Management", BCP 107, RFC 4107, June 2005.

[I-D.ietf-eap-keying] Aboba, B., Simon, D., and P. Eronen, "Extensible Authentication Protocol (EAP) Key Management Framework", draft-ietf-eap-keying-22 (work in progress), November 2007.

[I-D.ietf-hokey-reauth-ps] Clancy, C., Nakhjiri, M., Narayanan, V., and L. Dondeti, "Handover Key Management and Re-authentication Problem Statement", draft-ietf-hokey-reauth-ps-09 (work in progress), February 2008.

Appendix A. KDE Transport

A.1. KDE Transport over UDP

This section defines UDP transport of KDE. A well-known port number (to be assigned by IANA) is assigned for KDE.

For any KDE-PDU sent in response to another KDE-PDU received from the other peer, the source port is set to the well-known port number (to be assigned by IANA) assigned for KDE and the destination port is copied from the source port of the received KDE-PDU. For other KDE-PDUs, both the source and destination port numbers are set to the well-known port number (to be assigned by IANA) assigned for KDE.

A.2. KDE Transport over AAA

When KDE messages are carried in AAA protocols, they are carried in a RADIUS attribute or a corresponding Diameter AVP. The RADIUS attribute for KDE is defined as follows:

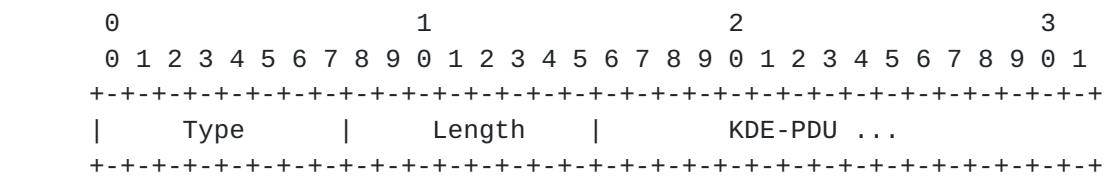

Figure 6: RADIUS Attribute for KDE

Type

IANA-TBD for KDE

Length

>=4

KDE-PDU

One KDE-PDU is included.

Authors' Addresses

Madjid Nakhjiri
Motorola

Email: madjid.nakhjiri@motorola.com

Yoshihiro Ohba
Toshiba

Email: yohba@tari.toshiba.com

Full Copyright Statement

Intellectual Property

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).