M. Thomas
J. Vilhuber
May 15, 2002

Kerberized Internet Negotiation of Keys (KINK)
draft-ietf-kink-kink-03.txt

Status of this Memo

Copyright Notice

Abstract

The Kerberized Internet Negotiation of Keys protocol (KINK) defines a low-latency, computationally inexpensive, easily managed, and cryptographically sound protocol to set up and maintain IPsec security associations using Kerberos authentication. KINK reuses many ISAKMP Quick Mode payloads to create, delete and maintain IPsec security associations which should lead to substantial reuse of existing IKE implementations.

Conventions used in this document

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.

1. Introduction

KINK is designed to provide a secure, scalable mechanism for establishing keys between communicating entities within a centrally managed environment in which it is important to maintain consistent security policy. The security goals of KINK are to provide privacy, authentication, and replay protection of key management messages, and to avoid denial of service vulnerabilities whenever possible. The performance goals of the protocol are to incur a low computational cost, to have low latency, to have a small footprint, and to avoid or minimize the use of public key operations. In particular, the protocol provides the capability to establish security associations in two messages with minimal computational effort.

Kerberos [KERB] and [KERBEROS] provides an efficient mechanism for trusted third party authentication for clients and servers. (Kerberos also provides an mechanisms for inter-realm authentication natively and with [PKCROSS].) Clients obtain tickets from an online authentication server (the Key Distribution Center or KDC). Tickets are then used to construct credentials for authenticating the client to the server. As a result of this authentication operation, the client and the server will also share a secret. KINK uses this property as the basis of distributing keys for IPsec.

The central key management provided by Kerberos is efficient because it limits computational cost and limits complexity versus IKE's necessity of using public key cryptography. Initial authentication to the KDC may be performed using either symmetric keys or asymmetric keys using [PKINIT]; however, subsequent requests for tickets, as well as authenticated exchanges between client and server always utilize symmetric cryptography. Therefore, public key operations (if any) are limited and are amortized over the lifetime of the initial authentication operation to the Kerberos KDC. For example, a client may use a single public key exchange with the KDC to efficiently establish multiple security associations with many other servers in the extended realm of the KDC. Kerberos also scales better than direct peer to peer keying when symmetric keys are used. The reason is that since the keys are stored in the KDC, the number of principal keys is O(n) rather than O(n*m), where "n" is the number of clients and "m" is the number of servers.

This document specifies the Kerberized Internet Negotiation of Keys Protocol and the domain of interpretation (DOI) for establishing and maintaining IPsec Security Associations [IPSEC]. No other domains of interpretation are defined in this document.

2. Terminology

Ticket
: A Kerberos term for a record that helps a client authenticate itself to a server; it contains the client's identity, a session

key, a lifetime, and other information, all sealed using the

server's secret key. The combination of a ticket and an authenticator (which proves freshness and knowledge of the key within the ticket) creates an authentication credential.

KDC
: Key Distribution Center, a network service that supplies tickets and temporary session keys; or an instance of that service or the host on which it runs. The KDC services both initial ticket and Ticket-Granting Ticket (TGT) requests. The initial ticket portion is referred to as the Authentication Server (or service). The Ticket-Granting Ticket portion is referred to as the Ticket-Granting Server (or service).

Realm
: A Kerberos administrative domain. A single KDC may be responsible for one or more realms. A fully qualified principal name includes a realm name along with a principal name unique within that realm.

TGT
: A ticket granting ticket is a normal Kerberos ticket which the KDC issues for the Kerberos service. The main purpose of a TGT is to capture the results of initial authentication for subsequent ticket granting requests, thus providing a single sign-on service.

User-User
: Kerberos normally divides the world into clients and servers where the server maintains a table of keys (keytab) which is used to encrypt/decrypt service tickets. In situations where a principal may not have a keytab (ex. a human/client principal rather than a service principal), Kerberos provides the means of issuing what is known as a User-User ticket. To produce the User-User ticket, the KDC requires the ticket granting tickets from both client principals. Kerberos does not specify a means obtaining a client's ticket granting ticket, and is thus application specific.

Principal
: Kerberos named entities are known as principals, and are roughly equivalent to X.509 distinguished names. Principals are either client or service principals. A principal is an entity that engages in a security relationship. A Kerberos principal name is

roughly equivalent to an X.509 distinguished name (it associates the principal with an adminsitrative domain). Principals may be client or servers. A server principal is generally distinguished by a flag in a KDC principal database and by a keytab maintained by the server.

DER
ASN.1 Distinguished Encoding Rules; Kerberos version 5 uses this encoding format of ASN.1.

Quick-Mode
IKE defines two phases: an authentication phase (phase 1, or Main Mode) and a security association maintenance phase (phase 2, or Quick Mode). KINK reuses IKE Quick Mode.

AP-REQ/AP-REP
Kerberos defines an standardized message format and transport for contacting a KDC to perform initial authentication, and for granting subsequent service tickets. When a client needs to authenticate to a server, Kerberos provides a standardized message format, but leaves the transport as application specific. The messages which perform this function are AP-REQ between the client and the server, and AP-REP between the server and client if mutual authentication is needed.

3. Protocol Overview

KINK is a command/response protocol which can create, delete and maintain IPsec security associations. Each command or response contains a common header along with a set of type-length-value payloads which are constrained according to the type of command or response. KINK itself is a stateless protocol in that each command or response does not require storage of hard state for KINK itself. This is in contrast to IKE's use of Main Mode to first establish an ISAKMP security association followed by subsequent Quick Mode exchanges.

KINK uses Kerberos mechanisms to provide mutual authentication, replay protection. For security association establishment. KINK provides privacy of the payloads which follow the Kerberos authenticator. KINK's design mitigates denial of service attacks by requiring authenticated exchanges before the use of any public key operations

and the installation of any state. KINK also provides the means of using Kerberos User-User mechanisms when there isn't a key shared between the server and the KDC. This is typically -- but not limited to -- the case with IPsec peers using [PKINIT] for initial authentication.

KINK directly reuses [ISAKMP] Quick Mode payloads, with some minor changes and omissions. In most cases, KINK exchanges are a single command and its response. The lone exception is the CREATE command which allows a final acknowledgment message when the respondent needs a full three-way handshake. This is only needed when the optimistic keying route is not taken, though it is expected that that will not be the norm. KINK also provides rekeying and dead peer detection as basic features.

4. Message Flows

KINK message flows all follow the same pattern between the two peers: a command, a response and a possible acknowledgment with CREATE's. The actual Kerberos KDC traffic here is for illustrative purposes only. In practice, when a principal obtains various tickets is a subject of Kerberos and local policy consideration. In these flows, we assume that A and B both have TGT's from their KDC.

4.1. Standard KINK Message Flow

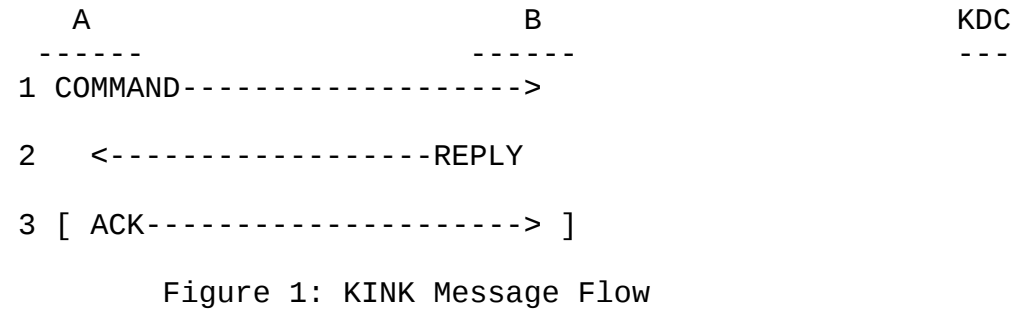


Figure 1: KINK Message Flow

4.2. GETTGT Message Flow

If the initiator determines that it will not be able to get a normal service ticket for the respondent (eg, B is a client principal), it MUST first fetch the TGT from the respondent in order to get a User-User service ticket:

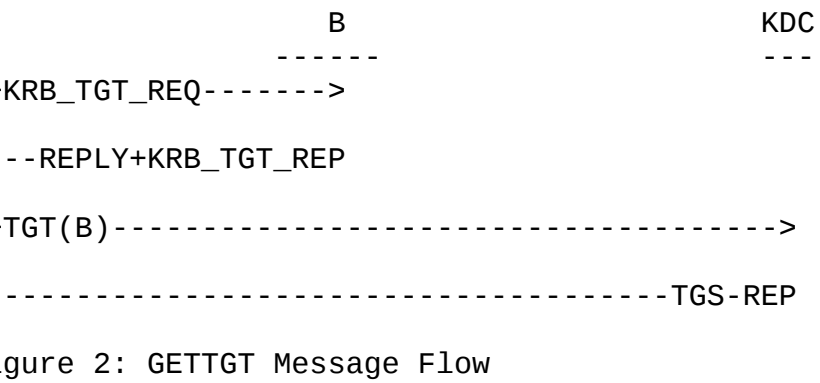
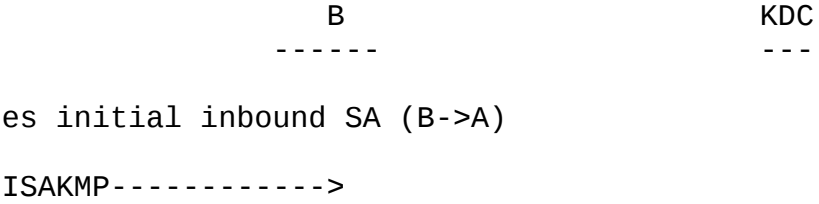


Figure 2: GETTGT Message Flow

4.3. CREATE Security Association

This flow instantiates a security association. The CREATE command takes an "optimistic" approach where security associations are initially created on the expectation that the respondent will chose the initial proposed payload. The optimistic payload is defined as the first transform of the first proposal of the first conjugate. The initiator MUST checks to see if the optimistic payload was selected by comparing all transforms and attributes which MUST be identical from the initiator's optimistic proposal with the lone exception of LIFE_KILOBYTES and LIFE_SECONDS. Both of these attributes MAY be set to a lower value by the respondent and still expect optimistic keying, but MUST NOT be set to a higher value which MUST generate an error.

CREATE'ing a security association on an existing SPI is an error in KINK and MUST be rejected with an ISAKMP notification of INVALID-SPI.



B creates inbound SA to A (A->B). If B chooses A's optimistic proposal, it creates the outbound SA as well (B->A).

2 <------------REPLY+ISAKMP

A creates outbound SA and modifies inbound SA if it first proposal wasn't acceptable.

3 [ACK-------------------->]

[B creates the outbound SA to A (B-A).]

Figure 3: CREATE Message Flow

The security associations are instantiated as follows: In step one host A creates an inbound security association in its security association database from B->A using the optimistic proposal in the ISAKMP SA proposal. It is then ready to receive any messages from B. A then sends the CREATE message to B. If B agrees to A's optimistic proposal, B instantiates a security association in its database from A->B. B then instantiates the security association from B->A. It then sends a REPLY to A without a NONCE payload and without requesting an ACK. If B does not choose the first proposal, it sends the actual choice in the REPLY, a NONCE payload and requests that the REPLY be acknowledged. Upon receipt of the REPLY, A modifies the inbound security association as necessary, instantiates the security association from A->B, If B requested an ACK, A now sends the ACK message. Upon receipt of the ACK, B installs the final security association from

B->A.

Note: if B adds a nonce, or does not choose the first proposal, it MUST request an ACK so that it can install the final outbound security association. The initiator MUST always generate an ACK if the ACKREQ bit is set in the KINK header, even if it believes that the respondent was in error.

4.3.1. CREATE Key Derivation Considerations

The CREATE command's optimistic approach allows a security association to be created in two messages rather than three. The implication of a two message exchange is that B will not contribute to the key since A must set up the inbound security association before it receives any additional keying material from B. Under normal circumstances this may be suspect, however KINK takes advantage of the fact that the KDC provides a reliable source of randomness which is

used in key derivation. In many cases, this will provide an adequate session key so that B will not require an acknowledgment. Since B is always at liberty to contribute to the keying material, this is strictly a key strength versus number of messages tradeoff which KINK implementations may decide as a matter of policy.

4.4. DELETE Security Association

The DELETE command deletes an existing security association. The DOI specific payloads describe the actual security association to be deleted. For the IPSEC DOI, those payloads will include an ISAKMP payload contains the SPI to be deleted in each direction.

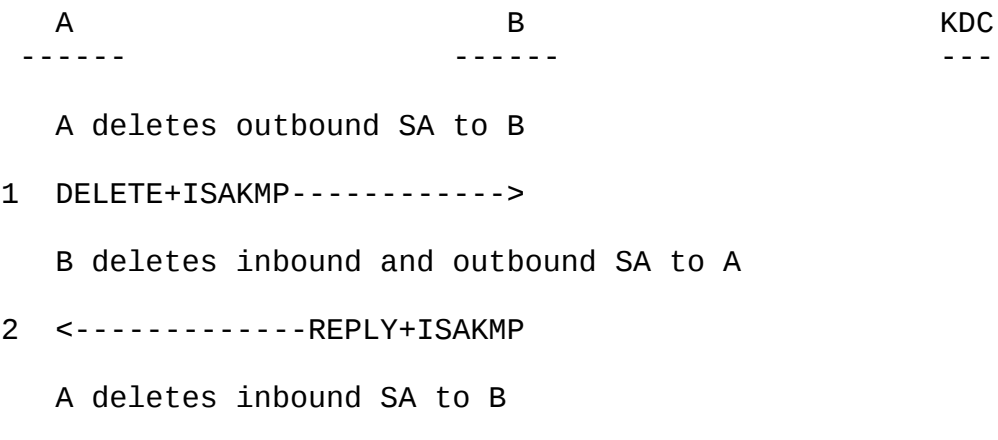


Figure 4: DELETE Message Flow

The DELETE command takes a "pessimistic approach" which does not delete incoming security associations until it receives acknowledgment that the other host has received the DELETE. The exception to the pessimistic approach is if the initiator wants to immediately

cease all activity on an incoming SA. In this case, it MAY delete the incoming SA as well in step one. If the receiver cannot find an appropriate SPI to delete, it MUST return an ISAKMP INVALID_SPI notification which also serves to inform the initiator that it can delete the incoming SA. For simplicity, KINK does not allow half open security associations; thus upon receiving a DELETE, the responder MUST delete its security associations, and MUST reply with ISAKMP delete notification messages if the SPI is found.

A race condition with DELETE exists. Packets in flight while the DELETE operation is taking place may, due to network reordering, etc,

arrive after the diagrams above recommend deleting the incoming security association. A KINK implementation SHOULD implement a grace timer which SHOULD be set to a period of at least two times the average round trip time, or to a configurable value. A KINK implementation MAY chose to set the grace period to zero at appropriate times to ungracefully delete a security association. The behavior described here loosely mimics the behavior of the TCP [RFC793] flags FIN and RST.

4.4.1. Rekeying Security Associations

KINK requires the initiator of a security association to be responsible for rekeying a security association. The reason is twofold: the first is to prevent needless duplication of security associations as the result of collisions due to an initiator and respondent both trying to renew an existing security association. The second reason is due to the client/server nature of Kerberos exchanges which expects the client to get and maintain tickets. While KINK requires that a KINK host be able to get and maintain tickets, in practice it is often advantageous for servers to wait for clients to initiate sessions so that they do not need to maintain a large ticket cache.

There are no special semantics for rekeying security associations in KINK. That is, in order to rekey an existing security association, the initiator must CREATE a new security association followed by either DELETE'ing the old security association or letting it time out. When identical flow selectors are available on different security associations, KINK implementations SHOULD choose the security association most recently created. It should be noted that KINK avoids most of the problems of [IKE] rekeying by having a reliable delete mechanism.

Normally a KINK implementation which rekeys existing security associations will try to rekey the security association ahead of a hard SA expiration. We call this time the rekey time Trekey. In order to avoid synchronization with similar implementations, KINK initiators MUST randomly pick a rekeying time between Trekey and the SA expiration time minus the amount of time it would take to go through a full retransmission time cycle, Tretrans. Trk SHOULD be set at least twice as high as Tretrans.

4.4.2. Dead Peer Detection

In order to determine that a KINK peer has lost its security database information, KINK peers MUST record the current epoch for which they

have valid SADB information and reflect that epoch in each AP-REQ and AP-REP message. When a KINK peer creates state for a given security association, it MUST also record the principal's epoch as well. If it discovers on a subsequent message that the principal's epoch has changed, it MUST consider all security associations created by that principal as invalid, and take some action such as tearing those SA's down.

It should be noted that KINK alone cannot provide a complete solution for dead peer detection since in many situations a KINK peer would have no reason to send subsequent KINK messages from whence it could determine the epoch mismatch. The larger picture may require some assistance from the IP layer itself to inform IPsec peers that they are sending SA protected data into a black hole. Assuming this mechanism is eventually defined, KINK peers SHOULD use this information as a hint that something is amiss and perform a dead peer detection operation by using the STATUS message to elicit a response which contains the peer's current epoch. Since the STATUS message is integrity protected, it in combination with network layer messaging should provide a secure means of recovery from dead peers.

4.5. STATUS Message Flow

At any point, a sender may send status, normally in the form of DOI specific payloads to its peer. In the case of the IPsec DOI, these are generally in the form of ISAKMP Notification Payloads.

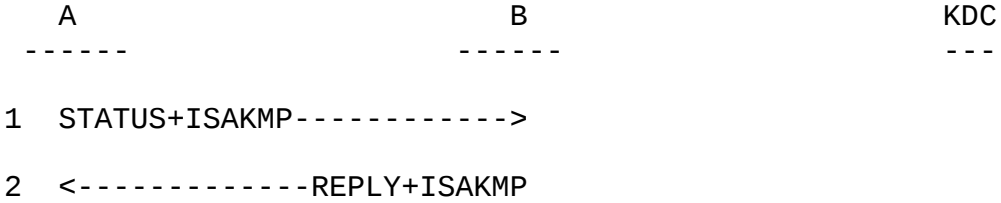


Figure 5: STATUS Message Flow

5. KINK Message Format

All values in KINK are formatted in network byte order: Most Significant Byte first.

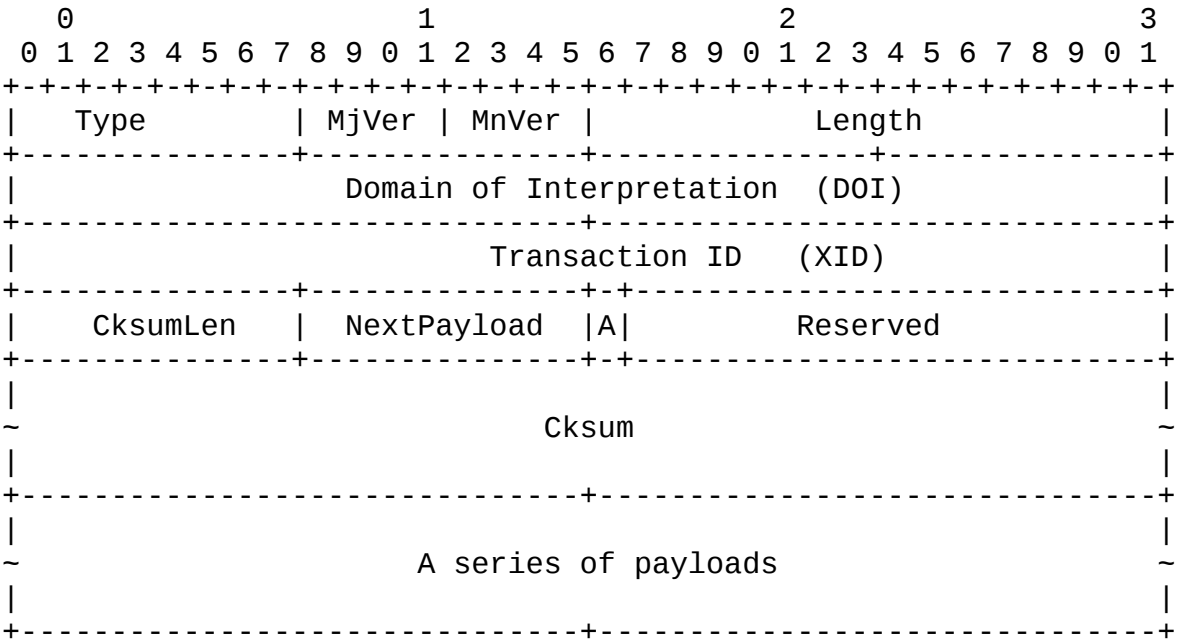


Figure 6: Format of a KINK message

Fields:

o Type (1 octet) - The type of message of this packet

Type	Value
RESERVED	0
CREATE	1
DELETE	2
REPLY	3
GETTGT	4
ACK	5
STATUS	6

o MjVer (4 bits) - Major protocol version number. This MUST be set to 1.

o MnVer (4 bits) - Minor protocol version number. This MUST be set to 0.

o Length (2 octets) - Length of the message in octets. Note that it is legal within KINK to omit the last bytes of padding in the last payload in the overall length.

o DOI (4 octets) - The domain of interpretation. All DOI's must be

registered with the IANA in the "Assigned Numbers" RFC [STD-2].

The IANA Assigned Number for the Internet IP Security DOI (IPSEC DOI) is one (1). This field defines the context of all other sub-payloads in this payloads. If other sub-payloads have a DOI field (example: Security Association Payload), then the DOI in that sub-payload MUST be checked against the DOI in this header, and the values MUST be the same.

- XID (4 octets) - The transaction ID. A KINK transaction is bound together by a transaction ID which is created by the command initiator and replicated in subsequent messages in the transaction. A transaction is defined as a command, a reply, and an optional acknowledgment. Transaction ID's are used by the initiator to discriminate between multiple outstanding requests to a respondent. It is not used for replay protection because that functionality is provided by Kerberos. The value of XID is chosen by the initiator and MUST be unique with all outstanding transactions. XID's MAY be constructed by using a monotonic counter, or random number generator.

- CksumLen (2 octets) -- CksumLen is the length in octets of the keyed hash of the message. A CksumLen of zero implies that the message is unauthenticated. Note that as with payload padding, the length here denotes the actual number of octets of the checksum structure not including any padding required.

- NextPayload (1 octet) -- Indicates the type of the first payload after the message header

- A (1 bit) -- ACK Request. Set to one if the responder requires an explicit acknowledgment that a REPLY was received. An initiator MUST NOT set this flag, nor should any other command other than CREATE request an ACK and then only when the optimistic SA is not chosen.

- Reserved (15 bits) -- Reserved and must be zero

- Cksum (variable) - Keyed checksum over the entire message. This field MUST always be present whenever a key is available via an AP-REQ or AP-REP payload. The key used MUST be the session key in the ticket. When a key is not available, this field is not present, and the CksumLen field is set to zero. The hash algorithm used is the same as specified in the etype for the Kerberos session key in the Kerberos ticket. If the etype does not specify a hash algorithm, SHA1 with a 20 byte checksum MUST be used. The format of the Cksum field is as follows:

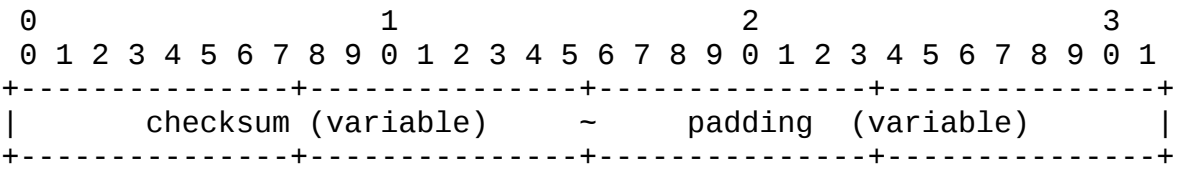


Figure 7: KINK Checksum

To compute the checksum, the checksum field is zeroed out and the

appropriate algorithm is run over the entire message (as given by the Length field in the KINK header), and placed in the Checksum field. To verify the checksum, the checksum is saved, and the checksum field is zeroed out. The checksum algorithm is run over the message, and the result is compared with the saved version. If they do not match, the message MUST be dropped.

The KINK header is followed immediately by a series of Type/Length/Value fields, defined in the next section.

5.1. KINK Payloads

Immediately following the header, there is a list of Type/Length/Value (TLV) payloads. There can be any number of payloads following the header. Each payload MUST begin with a payload header. Each payload header is built on the generic payload header. Any data immediately follows the generic header. Payloads are all implicitly padded to 4 octet boundaries, though the payload length field MUST accurately reflect the actual number of octets in the payload.

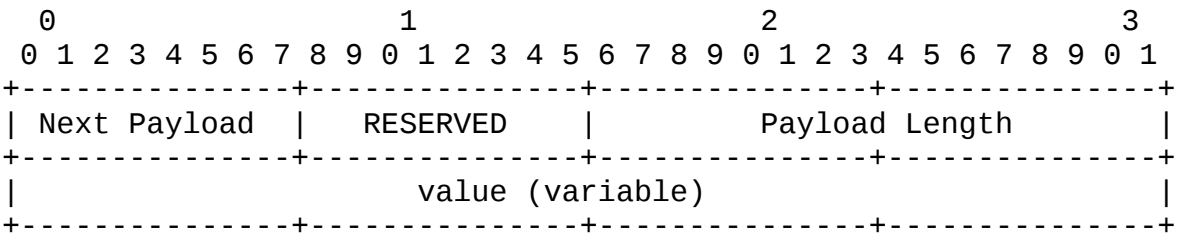


Figure 8: Format of a KINK payload

Fields:

o NextPayload (2 octets) - The type of the next payload

NextPayload Number

| ---- | ------ |
KINK_DONE	0 (same as ISAKMP_NEXT_NONE)
KINK_AP_REQ	KINK_ISAKMP_PAYLOAD_BASE+0
KINK_AP_REP	KINK_ISAKMP_PAYLOAD_BASE+1
KINK_KRB_ERROR	KINK_ISAKMP_PAYLOAD_BASE+2
KINK_TGT_REQ	KINK_ISAKMP_PAYLOAD_BASE+3
KINK_TGT_REP	KINK_ISAKMP_PAYLOAD_BASE+4
KINK_ISAKMP	KINK_ISAKMP_PAYLOAD_BASE+5
KINK_ENCRYPT	KINK_ISAKMP_PAYLOAD_BASE+6
KINK_ERROR	KINK_ISAKMP_PAYLOAD_BASE+7

NextPayload type KINK_DONE denotes that the current payload is the final payload in the message.

Note: the payload types are taken from the ISAKMP registry for

payload types. See the IANA consideration section for the value of KINK_ISAKMP_PAYLOAD_BASE.

- RESERVED (1 octet) - Unused, MUST be set to 0.
- Length (2 octets) - The length of this payload, including the Type and Length fields.
- Value (variable) - This value of this field depends on the Type.

5.1.1. KINK Padding Rules

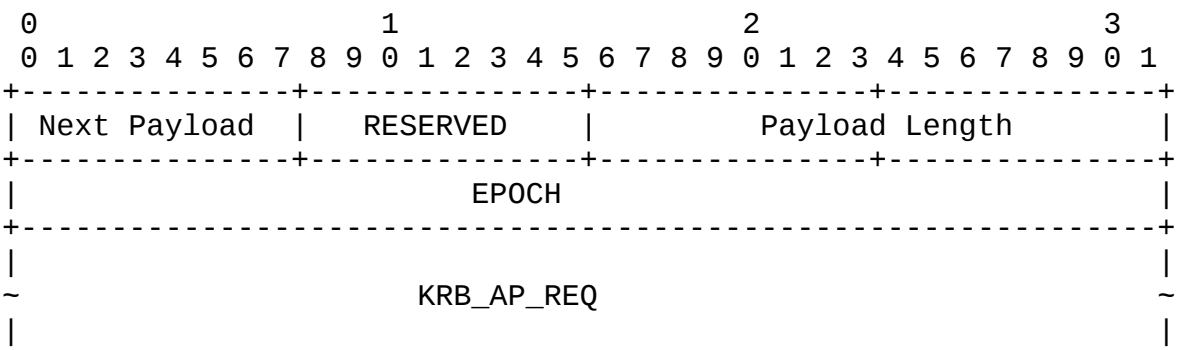
KINK has the following rules regarding alignment and padding:

- All length fields MUST reflect the actual number of octets in the structure; ie they do not account for padding bytes
- Between KINK payloads, checksums, headers or any other other variable length data, the adjacent fields MUST be aligned on 4 octet boundaries.
- Variable length fields MUST always start immediately after the last octet of the previous field. Ie, they are not padded to a 4 octet boundary.

5.1.2. 5.1.1 KINK_AP_REQ Payload

The KINK_AP_REQ payload relays a Kerberos AP-REQ to the respondent. The AP-REQ MUST request mutual authentication. The service that the KINK peer SHOULD request is "kink/fqdn@REALM" where "kink" is the KINK IPsec service, "fqdn" is the fully qualified domain name of the service host, and REALM is the Kerberos realm of the service. The exception to this rule is when User-User service is requested in which case the service name MUST be the service returned in the GetTGT response payload.

The value field of this payload has the following format:



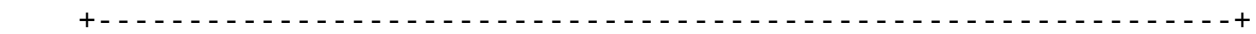


Figure 9: KINK_AP_REQ Payload

Fields:

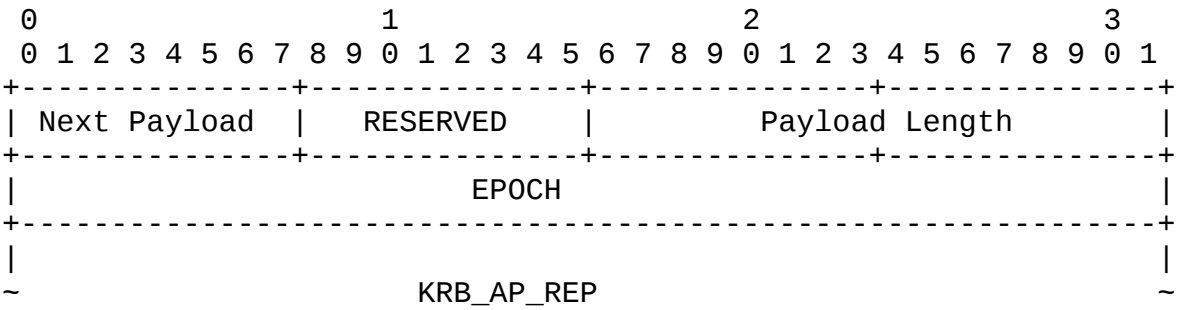
- EPOCH - the absolute time at which the creator of the AP-REQ has valid security database (SADB) information. Typically this is when the KINK keying daemon started if it does not retain SADB information across different restarts. The format of this fields is network order encoding of the standard posix four octet time stamp.

- KRB_AP_REQ - The value field of this payload contains a raw Kerberos KRB_AP_REQ.

5.1.3. KINK_AP_REP Payload

The KINK_AP_REP payload relays a kerberos AP-REP to the initiator. The AP-REP MUST be checked for freshness as described in [KERBEROS].

The value field of this payload has the following format:



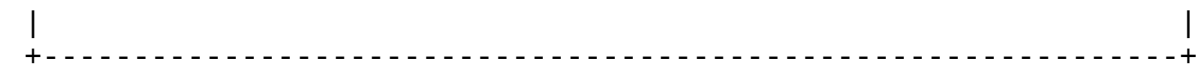


Figure 10: KINK_AP_REP Payload

Fields:

- EPOCH - the absolute time at which the creator of the AP-REP has valid security database (SADB) information. Typically this is when the KINK keying daemon started if it does not retain SADB information across different restarts. The format of this fields is network order encoding of the standard posix four octet time stamp.

- KRB_AP_REP - The value field of this payload contains a raw Kerberos KRB_AP_REP.

5.1.4. KINK_KRB_ERROR Payload

The KINK_KRB_ERROR payload relays Kerberos type errors back to the initiator. The receiver MUST be prepared to receive any valid [KERBEROS] error type, but the sender SHOULD send only the following errors:

KRB5KRB_AP_ERR_BAD_INTEGRITY
KRB5KRB_AP_ERR_TKT_EXPIRED
KRB5KRB_AP_ERR_SKEW

KRB5KRB_AP_ERR_NOKEY
KRB5KRB_AP_ERR_BADKEYVER

KINK implementations MUST make use of keyed Kerberos errors when the appropriate service key is available as specified in [KRBREVS]. In particular, clock skew errors MUST be integrity protected. For unauthenticated Kerberos errors, the receiver MAY choose to act on them, but SHOULD take precautions against make-work kinds of attacks.

Note that KINK does not make use of the text or e_data field of the Kerberos error message, though a compliant KINK implementation MUST be prepared to receive them and MAY log them.

The value field of this payload has the following format:

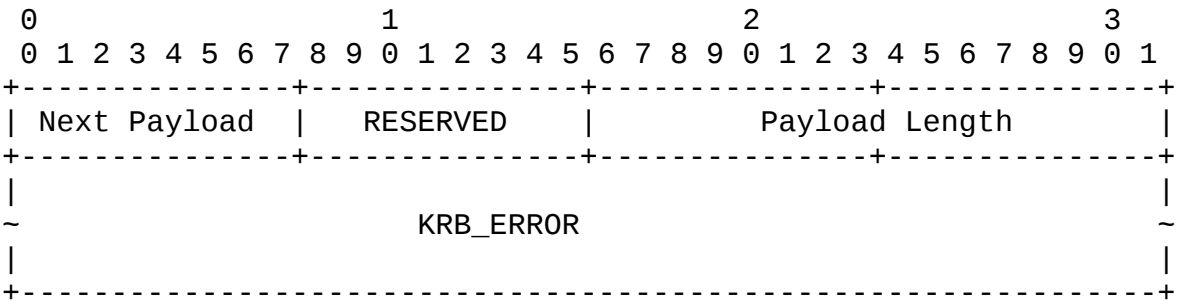


Figure 11: KINK_KRB_ERROR Payload

Fields:

o KRB_ERROR - The value field of this payload contains a raw Kerberos KRB_ERROR.

5.1.5. KINK_TGT_REQ Payload

The KINK_TGT_REQ payload provides a means to get a TGT from the peer

in order to obtain a User to User service ticket from the KDC

The value field of this payload has the following format:

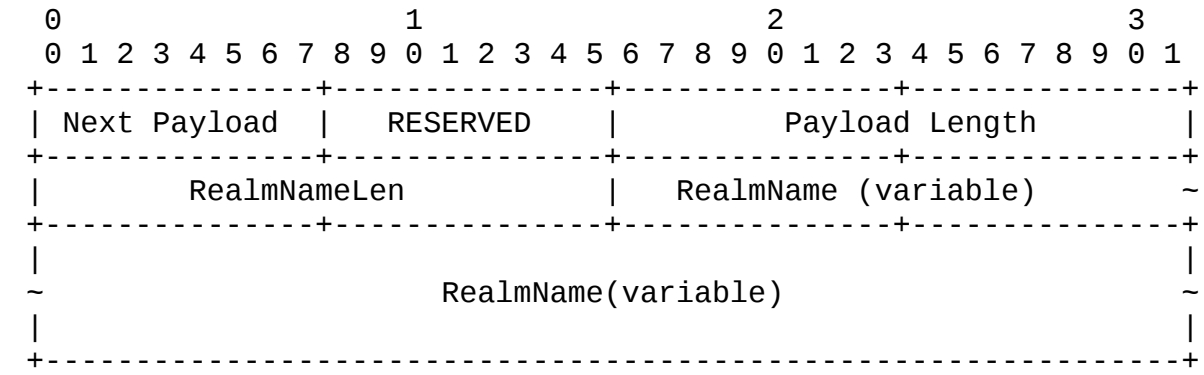


Figure 12: KINK_TGT_REQ Payload

Fields:

- RealmNameLen - The length of the realm name that follows
- RealmName - The realm name that the responder should return a TGT for.
- RESERVED - reserved and must be zero

If the responder is unable to get a TGT for the domain, it must reply with a KRB_ERROR payload type.

5.1.6. KINK_TGT_REP Payload

The value field of this payload contains the TGT requested in a previous KINK_TGT_REP command.

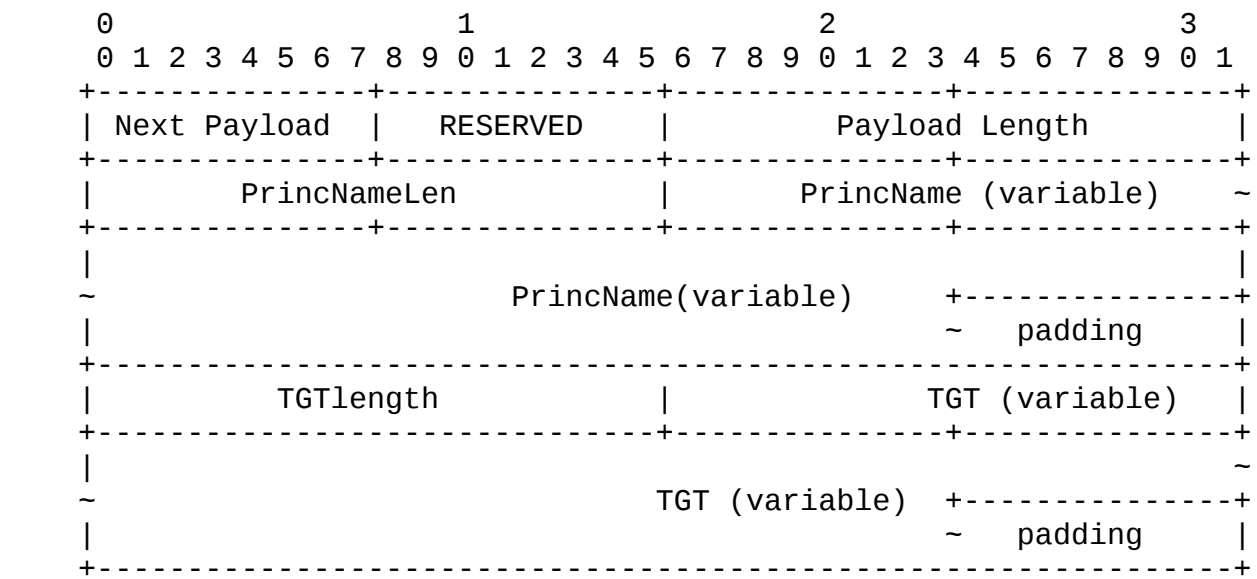


Figure 13: KINK_TGT_REQ Payload

Fields:

- PrincNameLen - The length of the principal name that immediately follows
- PrincName - The client principal that the initiator should request a User to User service ticket for.
- TGTlength - The length of TGT that immediately follows
- TGT - the DER encoded TGT of the responder

5.1.7. KINK_ISAKMP Payload

The value field of this payload has the following format:

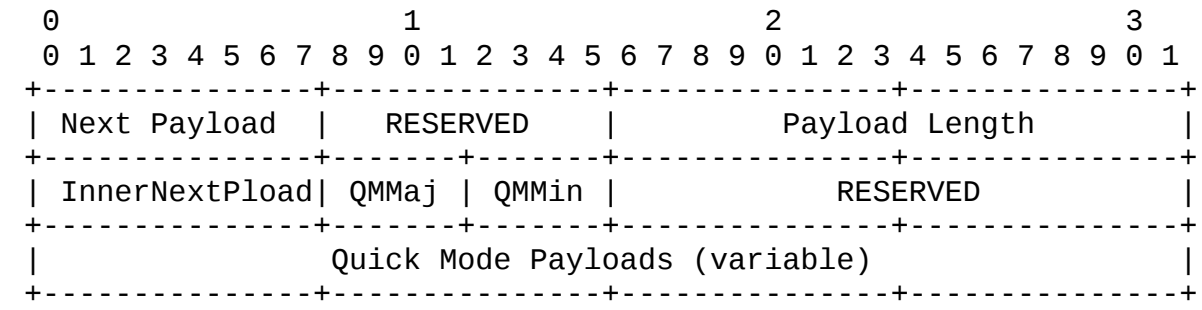


Figure 14: KINK_ISAKMP Payload

Fields:

- InnerNextPload - First payload type of the inner series of ISAKMP payloads.
- QMMaj - The major version of the inner payloads. MUST be set to 1.
- QMMin - The minor version of the inner payloads. MUST be set to 0.
- RESERVED - reserved and must be zero

  The KINK_ISAKMP payload encapsulates the IKE Quick Mode (phase two) payloads to take the appropriate action dependent on the KINK command. There may be any number of KINK_ISAKMP payloads within a single KINK message. While IKE is somewhat fuzzy about whether multiple different SA's may be created within a single IKE message, KINK explicitly requires that a new ISAKMP header be used for each discrete SA operation. In other words, a KINK sender MUST NOT send multiple quick mode transactions within a single KINK_ISAKMP payload.

  The purpose of the Quick Mode version is to allow backward compatibility with IKE and ISAKMP if there are subsequent revisions. At the present time, the Quick Mode major and minor versions are set to one and zero (1.0) respectively. These versions do not correspond to the ISAKMP version in the ISAKMP header. A compliant KINK implementation MUST support receipt of 1.0 payloads. It MAY support subsequent versions (both sending and receiving), and

SHOULD provide a means to resort back to Quick Mode version 1.0 if the KINK peer is unable to process future versions. A compliant KINK implementation MUST NOT mix Quick Mode versions in any given transaction.

5.1.8. KINK_ENCRYPT Payload

The KINK_ENCRYPT payload encapsulates other payloads and is encrypted using the encryption algorithm specified by the etype of the session key. This payload MUST be the final payload in the message. KINK encrypt payloads MUST be encrypted before the final KINK checksum is applied.

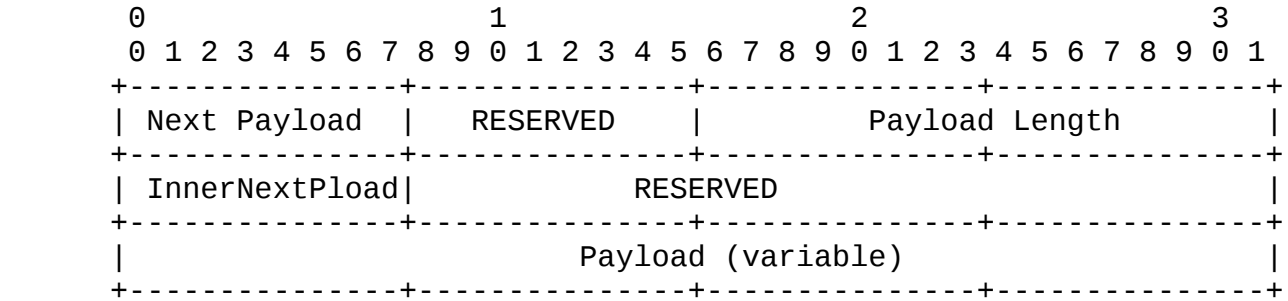


Figure 15: KINK_ENCRYPT Payload

Fields:

- InnerNextPload (variable) - First payload type of the inner series of encrypted KINK payloads.

- RESERVED - reserved and must be zero

  Note: the coverage of the encrypted data begins at InnerNextPload so that first payload's type is kept confidential. Thus, the number of encrypted octets is PayloadLength - 4.

  The format of the encryption payload uses the normal [KERBEROS] semantics of prepending a crypto-specific initialization vector and padding the entire message out to the crypto-specific number of bytes. For example, with DES-CBC, the initialization vector will be 8 octets long, and the entire message will be padded to an 8 octet boundary. Note that KINK Encrypt payload MUST NOT include

a checksum since this is redundant with the message integrity checksum in the KINK header.

5.1.9. KINK_ERROR Payload

The KINK_ERROR payload type provides a protocol level mechanism of returning an error condition. This payload should not be used for either Kerberos generated errors, or DOI specific errors which have their own payloads defined. The error code is in network order.

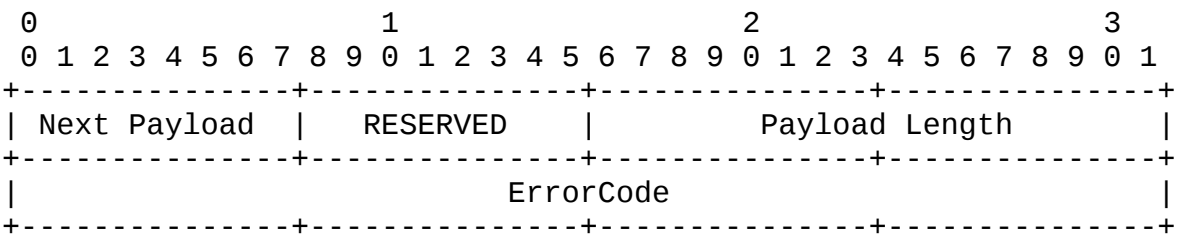


Figure 16: KINK_ERROR Payload

ErrorCode	Number	Purpose
KINK_OK	0	- No error detected
KINK_PROTOERR	1	- The message was malformed
KINK_INVDOI	2	- Invalid DOI
KINK_INVMAJ	3	- Invalid Major Version
KINK_INVMIN	4	- Invalid Minor Version
KINK_INTERR	5	- An unrecoverable internal error
RESERVED	6 - 8191	
Private Use	8192 - 16383	

6. KINK Quick Mode Payload Profile

KINK directly uses ISAKMP payloads to negotiate security associations. In particular, KINK uses IKE phase II payload types (aka Quick Mode). In general, there should be very few changes necessary to an IKE implementation to establish the security associations, and unless there is a note to the contrary in the memo, all capabilities and requirements in [IKE] MUST be supported. IKE Phase I payloads MUST NOT be sent.

Unlike IKE, KINK defines specific commands for creation, deletion, and status of security associations, mainly to facilitate predictable SA creation/deletion (see section 4.3 and 4.4). As such, KINK places certain restrictions on what payloads may be sent with which commands, and some additional restrictions and semantics of some of the payloads. Implementors should refer to [IKE] and [ISAKMP] for the actual format and semantics. If a particular IKE phase II payload is not mentioned here, it means that there are no differences in its use.

6.1. General Quick Mode Differences

- The Security Association Payload header for IP is defined in [IPDOI] section 4.6.1. For this memo, the Domain of Interpretation MUST be set to 1 (IPSec) and the Situation bitmap MUST be set to 1 (SIT_IDENTITY_ONLY). All other fields are omitted (because SIT_IDENTITY_ONLY is set).

- KINK also expands the semantics of IKE in it defines an optmistic proposal for CREATE commands to allow SA creation to complete in two messages.

- IKE Quick Mode (phase 2) uses the hash algorithm used in main mode (phase 1) to generate the keying material. KINK MUST use the hashing algorithm specified in the session ticket's etype.

- KINK does not use the HASH payload at all.

- KINK allows the NONCE payload Nr to be optional to facilitate optimistic keying.

6.2. Security Association Payloads

KINK supports the following security association attributes from [IPDOI]:

class	value	type
SA Life Type	1	B
SA Life Duration	2	V
Encapsulation Mode	4	B
Authentication Algorithm	5	B
Key Length	6	B
Key Rounds	7	B

Refer to [IPDOI] for the actual definitions for these attributes.

6.3. Proposal and Transform Payloads

KINK directly uses the Proposal and Transform payloads with no differences. KINK, however, places additional relevance to the first proposal and first transform of each conjugate for optimistic keying.

6.4. Identification Payloads

The Identification payload carries information that is used to identify the traffic that is to be protected using the keys exchanges in this memo. KINK restricts the ID types to the following values:

ID Type	Value
ID_IPV4_ADDR	1
ID_IPV4_ADDR_SUBNET	4
ID_IPV6_ADDR	5
ID_IPV6_ADDR_SUBNET	6
ID_IPV4_ADDR_RANGE	7
ID_IPV6_ADDR_RANGE	8

6.5. Nonce Payloads

The Nonce payload contains random data that MUST be used in key generation by the initiating KINK peer, and MAY be used by the responding KINK peer.

6.6. Notify Payloads

Notification information can be error messages specifying why an SA could not be established. It can also be status data that a process managing an SA database wishes to communicate with a peer process. For example, a secure front end or security gateway may use the Notify message to synchronize SA communication. The table below lists the Notification messages and their corresponding values that are supported in KINK.

NOTIFY MESSAGES - ERROR TYPES

Errors	Value
INVALID-PAYLOAD-TYPE	1
SITUATION-NOT-SUPPORTED	3
INVALID-MAJOR-VERSION	5
INVALID-MINOR-VERSION	6
INVALID-EXCHANGE-TYPE	7
INVALID-FLAGS	8
INVALID-MESSAGE-ID	9
INVALID-PROTOCOL-ID	10
INVALID-SPI	11
INVALID-TRANSFORM-ID	12
ATTRIBUTES-NOT-SUPPORTED	13
NO-PROPOSAL-CHOSEN	14
BAD-PROPOSAL-SYNTAX	15
PAYLOAD-MALFORMED	16
INVALID-KEY-INFORMATION	17
INVALID-ID-INFORMATION	18
ADDRESS-NOTIFICATION	26
NOTIFY-SA-LIFETIME	27
UNEQUAL-PAYLOAD-LENGTHS	30
RESERVED (Future Use)	31 - 8191
Private Use	8192 - 16383

NOTIFY MESSAGES - STATUS TYPES

Status	Value
CONNECTED	16384
RESERVED (Future Use)	16385 - 24575
DOI-specific codes	24576 - 32767
Private Use	32768 - 40959
RESERVED (Future Use)	40960 - 65535

6.7. Delete Payloads

KINK directly uses ISAKMP delete payloads with no changes.

6.8. KE Payloads

IKE requires that perfect forward secrecy be supported through the

use of the KE payload. However, Kerberos in general does not provide PFS so it is somewhat questionable whether a system which is heavily relying on Kerberos benefits from PFS. KINK retains the ability to use PFS, but relaxes the requirement from must implement to SHOULD implement.

7. IPsec DOI Message Formats

KINK messages are either commands, replies, or acknowledgments. A command is sent by an initiator to the respondent. A reply is sent by the respondent to the initiator. If the respondent desires confirmation of the reply, it sets the ACKREQ bit in the message header. The ACKREQ bit MUST NOT be set by the respondent except in the lone case of a CREATE message for which one of the security associations did not use the optimistic payload. In that case, the ACKREQ bit MUST be set. All commands, responses and acknowledgments are bound together by the XID field of the message header. The XID is normally a monotonically incrementing field, and is used by the initiator to differentiate between outstanding requests to a responder. The XID field does not provide replay protection as that functionality is provided by Kerberos mechanisms. In addition, commands and responses MUST use a cryptographic hash over the entire message if the two peers share a symmetric key via a ticket exchange.

7.1. REPLY Message Considerations

The REPLY message is a generic reply which MUST contain either a KINK_AP_REP, a KRB-ERROR, or KINK_ERROR payload. REPLY's MAY contain additional DOI specific payloads such as ISAKMP payloads which are defined in the following sections. The checksum in the KRB-ERROR message is not used, since the KINK header already contains a checksum field.

The server MUST return a KRB_AP_ERR_SKEW if the server clock and the client clock are off by more than the policy-determined clock skew limit (usually 5 minutes). The optional client's time in the KRB-ERROR MUST be filled out, and the client MUST compute the difference (in seconds) between the two clocks based upon the client and server time contained in the KRB-ERROR message. The client SHOULD store this clock difference and use it to adjust its clock in subsequent messages.

7.2. ACK Message Considerations

ACK's are sent only when the ACKREQ bit is set in a REPLY message. ACK's MUST NOT contain any payloads beside a lone AP-REQ header. If the initiator detects an error in the AP-REP or any other KINK or Kerberos error, it SHOULD take remedial action by reinitiating the initial command with the appropriate error to instruct the KINK receiver how to correct its original problem.

7.3. CREATE

This message initiates an establishment of new Security Association(s). The CREATE message must contain an AP-REQ payload and any DOI specific payloads.

```
CREATE KINK Header
  KINK_AP_REQ
  [KINK_ENCRYPT]
    KINK_ISAKMP payload
         SA Payload[s]
              Proposal Payloads
                       Transform Payloads
         Nonce Payload (Ni)
         [KE]
         [IDci, IDcr]
         [Notification Payloads]
```

Replies are of the following forms:

```
REPLY KINK Header
  KINK_AP_REP
  [KINK_ENCRYPT]
    KINK_ISAKMP
         SA Payload[s]
                Proposal Payload
                         Transform Payload
         [ Nonce Payload (Nr)]
         [IDci, IDcr]
         [Notification Payloads]
```

Note that there MUST be at least a single proposal payload and a single transform payload in REPLY messages. Also: unlike IKE, the Nonce Payload Nr is not required, and its absence means that the optimistic mode SA's installed by the initiator are valid. If any of the first proposals are not chosen by the recipient, it MUST include the nonce payload as well to indicate that the initiator's outgoing SA's must be modified.

KINK, like IKE allows the creation of many security associations in one create command. If any of the optimistic creation mode proposals is not chosen by the respondent, it MUST request an ACK.

If an IPspec DOI specific error is encountered, the respondent must reply with a Notify payload describing the error:

```
REPLY KINK Header
  KINK_AP_REP
  [KINK_ENCRYPT]
    [KINK_ERROR]
    KINK_ISAKMP
        [Notification Payloads]
```

If the respondent finds a Kerberos error for which it can produce a valid authenticator, the REPLY takes the following form:

```
REPLY KINK Header
  KINK_AP_REP
  [KINK_ENCRYPT]
    KINK_KRB_ERROR
```

Finally, if the respondent finds a Kerberos or KINK type of error it which it cannot create a AP-REP for, MUST reply with a lone KINK_KRB_ERROR or KINK_ERROR payload:

```
REPLY KINK Header
  [KINK_KRB_ERROR]
  [KINK_ERROR]
```

7.4. DELETE

This message indicates that the sending peer has deleted or will shortly delete Security Association(s) with the other peer.

```
DELETE KINK Header
  KINK_AP_REQ
  [KINK_ENCRYPT]
    [ KINK_ERROR payload ]
    KINK_ISAKMP payload
        Delete Payload[s]
        [Notification Payloads]
```

There are three forms of replies for a DELETE. The normal form is:

```
REPLY KINK Header
  KINK_AP_REP
  [KINK_ENCRYPT]
    [ KINK_ERROR payload ]
    KINK_ISAKMP payload
         Delete Payload[s]
         [Notification Payloads]
```


If an IPsec DOI specific error is encountered, the respondent must reply with a Notify payload describing the error:

```
REPLY KINK Header
  KINK_AP_REP payload
  [ KINK_ENCRYPT payload ]
    [ KINK_ERROR payload ]
    KINK_ISAKMP payload
        [Notification Payloads]
```

If the respondent finds a Kerberos error for which it can produce a valid authenticator, the REPLY takes the following form:

```
REPLY KINK Header
  KINK_AP_REP
  [KINK_ENCRYPT]
    KINK_KRB_ERROR
```

If the respondent finds a KINK or Kerberos type of error it MUST reply with a lone KINK_KRB_ERROR or KINK_ERROR payload:

```
REPLY KINK Header
  [KRB_ERROR]
  [KINK_KRB_ERROR]
```

7.5. STATUS

The STATUS command is used in two ways:

1) As a means to relay an ISAKMP Notification message

2) As a means of probing a peer whether its epoch has changed for dead peer detection.

```
      STATUS contains the following payloads:
        KINK Header
        KINK_AP_REQ payload
        [ [KINK_ENCRYPT]
            [ KINK_ERROR payload ]
            KINK_ISAKMP payload
               [Notification Payloads] ]
```


There are two forms of replies for a STATUS. The normal form is:

```
REPLY KINK Header
  KINK_AP_REP
  [ [KINK_ENCRYPT]
      [KINK_ERROR]
      KINK_ISAKMP
         [Notification Payloads] ]
```

If the respondent finds a Kerberos error for which it can produce a valid authenticator, the REPLY takes the following form:

```
REPLY KINK Header
  KINK_AP_REP
  [KINK_ENCRYPT]
    KINK_KRB_ERROR
```

If the respondent finds a KINK or Kerberos type of error it MUST reply with a lone KINK_KRB_ERROR or KINK_ERROR payload:

```
REPLY
  KINK Header
  [KRB_ERROR]
  [KINK_KRB_ERROR]
```

8. Key Derivation

KINK uses the same key derivation mechanisms that [IKE] uses in section 5.5, which is:

KEYMAT = prf(SKEYID_d, [g(qm)^xy |] protocol | SPI | Ni_b [| Nr_b])

The following differences apply:

- SKEYID_d is the session key in the Kerberos service ticket from the AP-REQ.

- Nr_b is optional

  By optional, it is meant that the equivalent of a zero length nonce was supplied.

  Note that g(qm)^xy refers to the keying material generated when KE payloads are supplied using Diffie Hellman key agreement. This is explained in section 5.5 of [IKE].

9. Transport Considerations

KINK uses UDP on port [XXX -- TBA by IANA] to transport its messages. There is one timer T which SHOULD take into consideration round trip

considerations and MUST implement a truncated exponential backoff mechanism. The state machine is simple: any message which expects a response MUST retransmit the request using timer T. Since Kerberos requires that messages be retransmitted with new times for replay protection, the message MUST be recreated each time including the checksum of the message. Both commands and replies with the ACKREQ bit set are kept on retransmit timers. When a KINK initiator receives a REPLY with the ACKREQ bit set, it MUST retain the ability to regenerate the ACK message for the transaction for a minimum of its a full retransmission timeout cycle or until it notices that packets have arrived on the newly constructed SA, whichever comes first.

When a KINK peer retransmits a message, it MUST create a new Kerberos authenticator for the AP-REQ so that the peer can differentiate between replays and dropped packets. This results in a potential race condition when a retransmission occurs before an in-flight reply is received/processed. To counter this race condition, the retransmitting party SHOULD keep a list of valid authenticators which are outstanding for any particular transaction.

10. Security Considerations

KINK cobbles together and reuses many parts of both Kerberos and IKE,

the latter which in turn is cobbled together from many other memos. As such, KINK inherits many of the weaknesses and considerations of each of its components. However, KINK uses only IKE Phase II payloads to create and delete security associations, the security considerations which pertain to IKE Phase I may be safely ignored.

KINK's use of Kerberos presents a couple of considerations. First, KINK explicitly expects that the KDC will provide adequate entropy when it generates session keys. Second, Kerberos is used as a user authentication protocol with the possibility of dictionary attacks on user passwords. This memo does not describe a particular method to avoid these pitfalls, but recommends that suitable randomly generated keys be used for the service principals such as using the -randomkey option with MIT's "kadmin addprinc" command as well as for clients when that is practical.

Kerberos itself does not provide for perfect forward secrecy which makes KINK's reliance on the IKE ability to do PFS somewhat suspect from an overall system's standpoint. In isolation KINK itself should be secure from offline analysis from compromised principal passphrases if PFS is used, but the existence of other Kerberized service which do not provide PFS makes this a less than optimal situation on the whole.

10.1. Security Policy Database Considerations

KINK leaves the population of the IPsec security policy database out of scope. There are, however, considerations which should be pointed

out. First, even though when and when not to initiate a user to user flow is left to the discretion of the KINK implementation, a Kerberos client which initially authenticated using a symmetric key SHOULD NOT use a user-user flow if the respondent is also in the same realm. Likewise, a KINK initiator which authenticated in a public key realm SHOULD use a user-user flow if the respondent is in the same realm.

At a minimum the security policy database for a KINK implementation SHOULD contain a logical record of the KDC to contact, principal name for the respondent, and whether the KINK implementation should use a direct AP-REQ/AP-REP flow, or a User-User flow to CREATE/DELETE the security association.

That said, there is considerable room for improvement on how a KINK initiator could auto-discover how a respondent in a different realm initially authenticated. This is left as an implementation detail as well as the subject of possible future standardization efforts which are outside of the scope of the KINK working group.

11. IANA Considerations

KINK requires that a new well known system port for UDP be created. Since KINK uses standard message types from [IKE], KINK does not require any new registries. Any new DOI's, ISAKMP types, etc for future versions of KINK MUST use the registries defined for [IKE].

In addition, the ISAKMP payload types currently don't have a IANA registry, but needs one. KINK defines its payload constants as a sequential set of integers from KINK_ISAKMP_PAYLOAD_BASE to KINK_ISAKMP_PAYLOAD_BASE+7.

12. Related Work

The IPsec working group has defined a number of protocols that provide the ability to create and maintain cryptographically secure security associations at layer three (ie, the IP layer). This effort has produced two distinct protocols:

- a mechanism for encrypting and authenticating IP datagram payloads which assumes a shared secret between the sender and receiver
- a mechanism for IPsec peers to perform mutual authentication and exchange keying material

The IPsec working group has defined a peer to peer authentication and keying mechanism, IKE (RFC 2409). One of the drawbacks of a peer to peer protocol is that each peer must know and implement a site's

security policy which in practice can be quite complex. In addition, the peer to peer nature of IKE requires the use of Diffie Hellman (DH) to establish a shared secret. DH, unfortunately, is computationally quite expensive and prone to denial of service attacks. IKE also relies on X.509 certificates to realize scalable authentication of peers. Digital signatures are also computationally expensive and certificate based trust models are difficult to deploy in practice. While IKE does allow for pre-shared symmetric keys, key distribution is required between all peers -- an O(n2) problem -- which is problematic for large deployments.

13. Normative References

[KERBEROS]
J. Kohl, C. Neuman. The Kerberos Network Authentication Service (V5). Request for Comments 1510.

[IPSEC]
S. Kent, R. Atkinson. Security Architecture for the Internet Protocol. Request for Comments 2401.

[IKE]D. Harkins, D. Carrel. The Internet Key Exchange (IKE). Request for Comments 2409.

[ISAKMP]
Maughhan, D., Schertler, M., Schneider, M., and J. Turner, "Internet Security Association and Key Management Protocol (ISAKMP)", RFC 2408, November 1998.

[IPDOI]
Piper, D., "The Internet IP Security Domain Of Interpretation for ISAKMP", RFC 2407, November 1998.

14. Informative References

[RFC2412]
Orman, H., "The OAKLEY Key Determination Protocol", RFC 2412, November 1998.

[RFC793]

Postel, J., "Transmission Control Protocol", RFC 793, Sep-01-1981

[KERB]B.C. Neuman, Theodore Ts'o. Kerberos: An Authentication Service for Computer Networks, IEEE Communications, 32(9):33-38. September 1994.

[PKINIT]
B. Tung, C. Neuman, M. Hur, A. Medvinsky, S.Medvinsky, J. Wray, J. Trostle. Public Key Cryptography for Initial Authentication in Kerberos. draft-ietf-cat-kerberos-pk-init-11.txt

[PKCROSS]
M.Hur, B. Tung, T. Ryutov, C. Neuman, G. Tsudik, A. Medvinsky, B. Sommerfeld. Public Key Cryptography for Cross-Realm Authentication in Kerberos. draft-ietf-cat-kerberos-pk-cross-06.txt

15. Mailing List

Please send comments to the KINK mailing list (ietf-kink@vpnc.org). You can subscribe by sending mail to ietf-kink-request@vpnc.org with a line in the body of the mail with the word SUBSCRIBE in it.

16. Author's Addresses

Michael Thomas
Jan Vilhuber
Cisco Systems
170 West Tasman Drive
San Jose, CA 95134
E-mail: {mat,vilhuber}@cisco.com

17. Acknowledgments

Many have contributed to the KINK effort, including our working group chairs Derek Atkins and Jonathan Trostle. The original inspiration came from Cablelab's Packetcable effort which defined a simplifed version of Kerberized IPsec, including Sasha Medvinsky, Mike Froh, and Matt Hur and David McGrew. The inspiration for wholly reusing IKE Phase II is the result of the Tero Kivinen's draft suggesting grafting Kerberos authentication onto quick mode.