Network Working Group
Request for Comments: 4430
Category: Standards Track

S. Sakane
K. Kamada
Yokogawa Electric Corp.
M. Thomas
J. Vilhuber
Cisco Systems

March 2006

# Kerberized Internet Negotiation of Keys (KINK)

## Status of This Memo



## Copyright Notice



## Abstract

This document describes the Kerberized Internet Negotiation of Keys (KINK) protocol. KINK defines a low-latency, computationally inexpensive, easily managed, and cryptographically sound protocol to establish and maintain security associations using the Kerberos authentication system. KINK reuses the Quick Mode payloads of the Internet Key Exchange (IKE), which should lead to substantial reuse of existing IKE implementations.

## Table of Contents

## 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 have a low computational cost, low latency, and a small footprint. It is also to avoid or minimize the use of public key operations. In particular, the protocol provides the capability to establish IPsec security associations (SAs) in two messages with minimal computational effort. These requirements are described in RFC 3129 [REQ4KINK].

Kerberos [KERBEROS] provides an efficient authentication mechanism for clients and servers using a trusted third-party model. Kerberos also provides a mechanism for cross-realm authentication natively. A client obtains a ticket from an online authentication server, the Key Distribution Center (KDC). The ticket is then used to construct a credential for authenticating the client to the server. As a result of this authentication operation, the server will also share a secret key with the client. 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 [IKE]. Initial authentication to the KDC may be performed using either symmetric keys, or asymmetric keys using the Public Key Cryptography for Initial Authentication in Kerberos [PKINIT]; however, subsequent requests for tickets as well as authenticated exchanges between the client and servers always utilize symmetric cryptography. Therefore, public key operations (if any) are limited and are amortized over the lifetime of the credentials acquired in the initial authentication operation to the KDC. For example, a client may use a single public key exchange with the KDC to efficiently establish multiple SAs with many other servers in the 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+m) rather than O(n*m), where "n" is the number of clients and "m" is the number of servers.

### 1.1. 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 [RFC2119].

It is assumed that the readers are familiar with the terms and concepts described in Kerberos Version 5 [KERBEROS], IPsec [IPSEC], and IKE [IKE].

## 2. Protocol Overview

KINK is a command/response protocol that can create, delete, and maintain IPsec SAs. Each command or response contains a common header along with a set of type-length-value payloads. The type of a command or a response constrains the payloads sent in the messages of the exchange. KINK itself is a stateless protocol in that each command or response does not require storage of hard state for KINK. This is in contrast to IKE, which uses Main Mode to first establish an Internet Security Association and Key Management Protocol (ISAKMP) SA followed by subsequent Quick Mode exchanges.

KINK uses Kerberos mechanisms to provide mutual authentication and replay protection. For establishing SAs, KINK provides confidentiality for the payloads that follow the Kerberos AP-REQ payload. The design of KINK 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 a means of using Kerberos User-to-User mechanisms when there is not 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 Quick Mode payloads defined in section 5.5 of [IKE], with some minor changes and omissions. In most cases, KINK exchanges are a single command and its response. An optional third message is required when creating SAs, only if the responder rejects the first proposal from the initiator or wants to contribute the keying materials. KINK also provides rekeying and dead peer detection.

## 3. Message Flows

All KINK message flows follow the same pattern between the two peers: a command, a response, and an optional acknowledgement in a CREATE flow. A command is a GETTGT, CREATE, DELETE, or STATUS message; a response is a REPLY message; and an acknowledgement is an ACK message.

KINK uses Kerberos as the authentication mechanism; therefore, a KINK host needs to get a service ticket for each peer before actual key negotiations. This is basically a pure Kerberos exchange and the actual 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. As an exception, the GETTGT message flow of KINK (described in section 3.1) is used when a User-to-User authentication is required. In this flow, we assume that both A and B have ticket-granting tickets (TGTs) from their KDCs.

After a service ticket is obtained, KINK uses the CREATE message flow (section 3.2), DELETE message flow (section 3.3), and STATUS message flow (section 3.4) to manage SAs. In these flows, we assume that A has a service ticket for B.

## 3.1. GETTGT Message Flow

This flow is used to retrieve a TGT from the remote peer in User-to-User authentication mode.

If the initiator determines that it will not be able to get a normal (non-User-to-User) service ticket for the responder, it can try a User-to-User authentication. In this case, it first fetches a TGT from the responder in order to get a User-to-User service ticket:

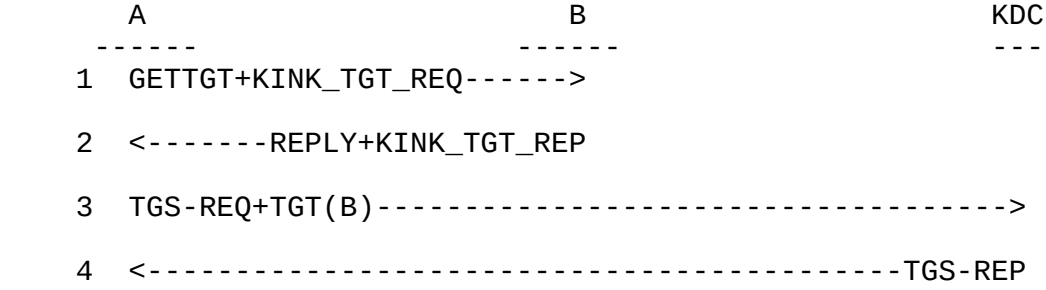


Figure 1: GETTGT Message Flow

The initiator MAY support the following events as triggers to go to the User-to-User path. Note that the two errors described below will not be authenticated, and how to act on them depends on the policy.

- The local policy says that the responder requires a User-to-User authentication.
- A KRB_AP_ERR_USER_TO_USER_REQUIRED error is returned from the responder.
- A KDC_ERR_MUST_USE_USER2USER error is returned from the KDC.

### 3.2. CREATE Message Flow

This flow creates SAs. The CREATE command takes an "optimistic" approach, where SAs are initially created on the expectation that the responder will choose the initial proposed payload. The optimistic proposal is placed in the first transform payload(s) of the first proposal. The initiator MUST check to see if the optimistic proposal was selected by comparing all transforms and attributes, which MUST be identical to those in the initiator's optimistic proposal with the exceptions of LIFE_KILOBYTES and LIFE_SECONDS. Each of these attributes MAY be set to a lower value by the responder and still expect optimistic keying, but MUST NOT be set to a higher value that MUST generate a NO-PROPOSAL-CHOSEN error. The initiator MUST use the shorter lifetime.

When a CREATE command contains an existing Security Parameter Index (SPI), the responder MUST reject it and SHOULD return an ISAKMP notification with INVALID-SPI.

When a key exchange (KE) payload is sent from the initiator but the responder does not support it, the responder MUST reject it with an ISAKMP notification of INVALID-PAYLOAD-TYPE containing a KE payload type as its notification data. When the initiator receives this error, it MAY retry without a KE payload (as another transaction) if its policy allows that.

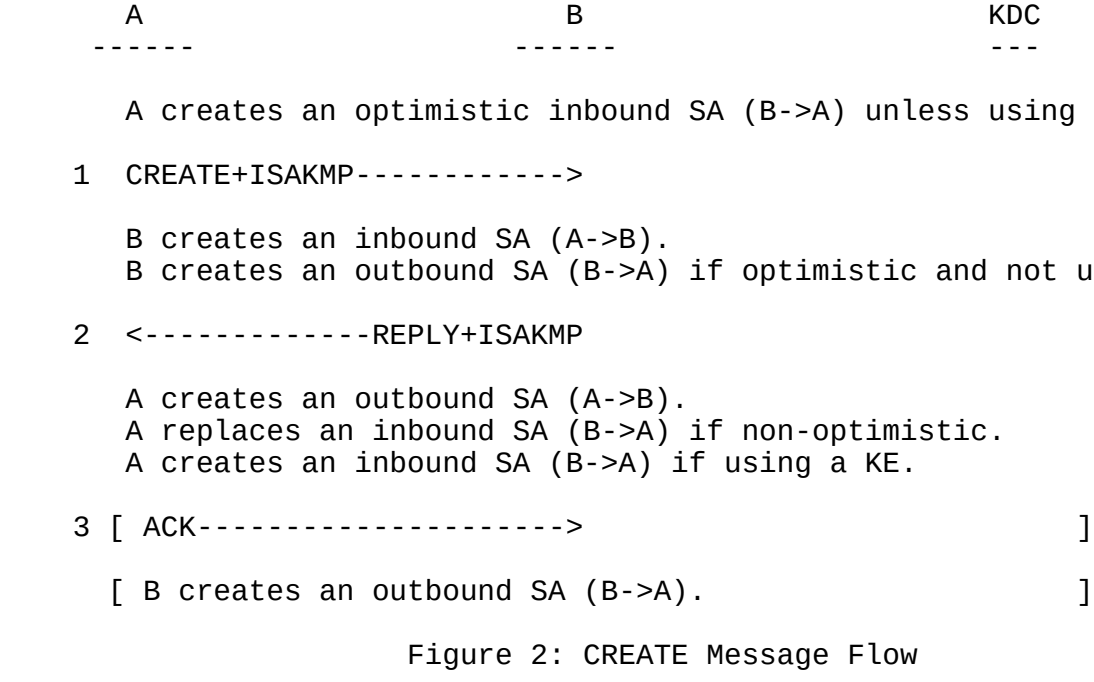


Figure 2: CREATE Message Flow

Creating SAs has two modes: 2-way handshake and 3-way handshake. The initiator usually begins a negotiation expecting a 2-way handshake. When the optimistic proposal is not chosen by the responder, the negotiation is switched to a 3-way handshake. When and only when the initiator uses a KE payload, 3-way handshake is expected from the beginning.

A 2-way handshake is performed in the following steps:

1) The host A creates an inbound SA (B->A) in its SA database using the optimistic proposal in the ISAKMP SA proposal. It is then ready to receive any messages from B.
2) A then sends the CREATE message to B.
3) If B agrees to A's optimistic proposal, B creates an inbound SA (A->B) and an outbound SA (B->A) in its database. If B does not choose the first proposal or wants to add a Nonce payload, switch to step 3 of the 3-way handshake described below.
4) B then sends a REPLY to A without a Nonce payload and without requesting an ACK.
5) Upon receipt of the REPLY, A creates an outbound SA (A->B).

A 3-way handshake is performed in the following steps:

1) The host A sends the CREATE message to B without creating any SA.
2) B chooses one proposal according to its policy.
3) B creates an inbound SA (A->B) and sends the actual choice in the REPLY. It SHOULD send the optional Nonce payload (as it does not increase message count and generally increases entropy sources) and MUST request that the REPLY be acknowledged.
4) Upon receipt of the REPLY, A creates the inbound SA (B->A) (or modifies it as necessary, if switched from 2-way), and the outbound SA (A->B).
5) A now sends the ACK message.
6) Upon receipt of the ACK, B installs the final outbound SA (B->A).

If B does not choose the first proposal, adds a nonce, or accepts the KE exchange, then it MUST request an ACK (i.e., set the ACKREQ bit) so that it can install the final outbound SA. The initiator MUST always generate an ACK if the ACKREQ bit is set in the KINK header, even if it believes that the responder was in error.

### 3.2.1. CREATE Key Derivation Considerations

The CREATE command's optimistic approach allows an SA 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 SA before it receives any additional keying material from B. This may be suspect under normal circumstances; 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 acknowledgement. Since B is always at liberty to contribute to the keying material, this is strictly a trade-off between the key strength versus the number of messages, which KINK implementations may decide as a matter of policy.

### 3.3. DELETE Message Flow

The DELETE command deletes existing SAs. The domain of interpretation (DOI)-specific payloads describe the actual SA to be deleted. For the IPsec DOI, those payloads will include an ISAKMP payload containing the list of the SPIs to be deleted.

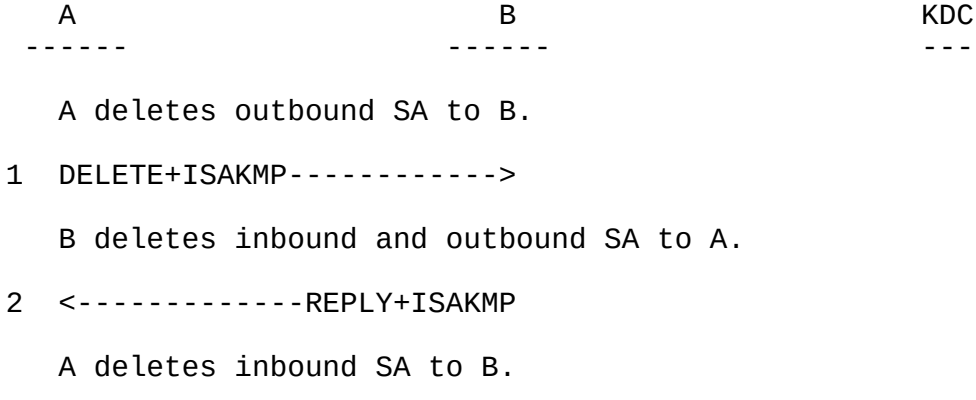


Figure 3: DELETE Message Flow

The DELETE command takes a "pessimistic" approach, which does not delete inbound SAs until it receives acknowledgement 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 inbound SA. In this case, it MAY delete the inbound SA as well in step 1, above.

The ISAKMP payload contains ISAKMP Delete payload(s) that indicate the inbound SA(s) for the initiator of this flow. KINK does not allow half-open SAs; thus, when the responder receives a DELETE command, it MUST delete SAs of both directions, and MUST reply with ISAKMP Delete payload(s) that indicate the inbound SA(s) for the responder of this flow. If the responder cannot find an appropriate SPI to be deleted, it MUST return an ISAKMP notification with INVALID_SPI, which also serves to inform the initiator that it can delete the inbound SA.

A race condition with the DELETE flow exists. Due to network reordering, etc., packets in flight while the DELETE operation is taking place may arrive after the diagrams above, which recommend deleting the inbound SA. A KINK implementation SHOULD implement a grace timer that 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 choose to set the grace period to zero at appropriate times to delete an SA ungracefully. The behavior described here is referred from the behavior of the TCP [RFC793] flags FIN and RST.

## 3.4. STATUS Message Flow

This flow is used to send any information to a peer or to elicit any information from a peer. An initiator may send a STATUS command to the responder at any time, optionally with DOI-specific ISAKMP payloads. In the case of the IPsec DOI, these are generally in the form of ISAKMP Notification payloads. A STATUS command is also used as a means of dead peer detection described in section 3.7.

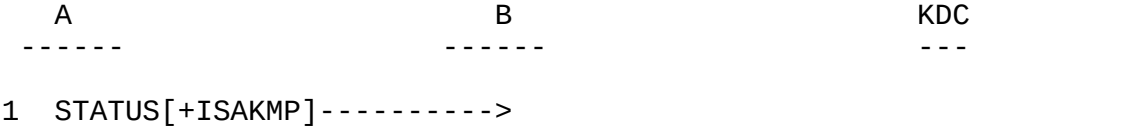


Figure 4: STATUS Message Flow

## 3.5. Reporting Errors

When the responder detects an error in a received command, it can send a DOI-specific payload to indicate the error in a REPLY message. There are three types of payloads that can indicate errors: KINK_KRB_ERROR payloads for Kerberos errors, KINK_ERROR payloads for KINK errors, and KINK_ISAKMP payloads for ISAKMP errors. Details are described in sections 4.2.3, 4.2.8, and 4.2.6, respectively.

If the initiator detects an error in a received reply, there is no means to report it back to the responder. The initiator SHOULD log the event and MAY take a remedial action by reinitiating the initial command.

If the server clock and the client clock are off by more than the policy-determined clock skew limit (usually 5 minutes), the server MUST return a KRB_AP_ERR_SKEW. The optional client's time in the KRB-ERROR SHOULD be filled out. If the server protects the error by adding the Cksum field and returning the correct client's time, the

client SHOULD 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. If the error is not protected, the client MUST NOT use the difference to adjust subsequent messages, because doing so would allow an attacker to construct authenticators that can be used to mount replay attacks.

### 3.6. Rekeying Security Associations

KINK expects the initiator of an SA to be responsible for rekeying the SA for two reasons. The first reason is to prevent needless duplication of SAs as the result of collisions due to an initiator and responder both trying to renew an existing SA. The second reason is due to the client/server nature of Kerberos exchanges, which expects the client to get and maintain tickets. While KINK expects that a KINK host is 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 SAs in KINK. That is, in order to rekey an existing SA, the initiator must CREATE a new SA followed by either deleting the old SA with the DELETE flow or letting it time out. When identical flow selectors are available on different SAs, KINK implementations SHOULD choose the SA 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 that rekeys existing SAs will try to rekey the SA ahead of an SA termination, which may include the hard lifetime in time/bytecount or the overflow of the sequence number counter. We call this time "soft lifetime". The soft lifetime MUST be randomized to avoid synchronization with similar implementations. In the case of the lifetime in time, one reasonable approach to determine the soft lifetime is picking a random time between T-rekey and T-retrans and subtracting it from the hard lifetime. Here, T-rekey is the reasonable maximum rekeying margin, and T-retrans is the amount of time it would take to go through a full retransmission cycle. T-rekey SHOULD be at least twice as high as T-retrans.

### 3.7. 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 SA information for a peer and reflect that epoch in each AP-REQ and AP-REP message. When a KINK peer creates state for a given SA, it MUST also record the principal’s epoch. If it discovers

on a subsequent message that the principal's epoch has changed, it MUST consider all SAs created by that principal as invalid, and take some action such as tearing those SAs down.

While a KINK peer SHOULD use feedback from routing (in the form of ICMP messages) as a trigger to check whether or not the peer is still alive, a KINK peer MUST NOT conclude the peer is dead simply based on unprotected routing information (said ICMP messages).

If there is suspicion that a peer may be dead (based on any information available to the KINK peer, including lack of IPsec traffic, etc.), the KINK STATUS message SHOULD be used to coerce an acknowledgement out of the peer. Since nothing is negotiated about dead peer detection in KINK, each peer can decide its own metric for "suspicion" and also what timeouts to use before declaring a peer dead due to lack of response to the STATUS message. This is desirable, and does not break interoperability.

The STATUS message has a twofold effect. First, it elicits a cryptographically secured (and replay-protected) response from the peer, which tells us whether or not the peer is reachable/alive. Second, it carries the epoch number of the peer, so we know whether or not the peer has rebooted and lost all state. This is crucial to the KINK protocol: In IKE, if a peer reboots, we lose all cryptographic context, and no cryptographically secure communication is possible without renegotiating keys. In KINK, due to Kerberos tickets, we can communicate securely with a peer, even if the peer rebooted, as the shared cryptographic key used is carried in the Kerberos ticket. Thus, active cryptographic communication is not an indication that the peer has not rebooted and lost all state, and the epoch is needed.

Assume a Peer A sending a STATUS and a peer B sending the REPLY (see section 3.4). Peer B MAY assume that the sender is alive, and the epoch in the STATUS message will indicate whether or not the peer A has lost state. Peer B MUST acknowledge the STATUS message with a REPLY message, as described in section 3.4.

The REPLY message will indicate to peer A that the peer is alive, and the epoch in the REPLY will indicate whether peer B has lost its state or not. If peer A does not receive a REPLY message from peer B in a suitable timeout, peer A MAY send another STATUS message. It is up to peer A to decide how aggressively to declare peer B dead. The level of aggressiveness may depend on many factors such as rapid fail over versus number of messages sent by nodes with large numbers of SAs.

Note that peer B MUST NOT make any inferences about a lack of STATUS message from peer A. Peer B MAY use a STATUS message from peer A as an indication of A's aliveness, but peer B MUST NOT expect another STATUS message at any time (i.e., dead peer detection is not periodic keepalives).

Strategies for sending STATUS messages are the following: Peer A may decide to send a STATUS message only after a prolonged period where no traffic was sent in either direction over the IPsec SAs with the peer. Once there is traffic, peer A may want to know if the traffic is going into a black hole, and send a STATUS message. Alternatively, peer A may use an idle timer to detect lack of traffic with the peer, and send STATUS messages in the quiet phase to make sure the peer is still alive for when traffic needs to finally be sent.

### 3.7.1. Coping with Dead User-to-User Peers

When an initiator uses a User-to-User ticket and a responder has lost its previous TGT, the usual dead peer detection (DPD) mechanism does not work, because the responder cannot decrypt the ticket with its new TGT. In this case, the following actions are taken.

- When the responder receives a KINK command with a User-to-User ticket that cannot be decrypted with its TGT, it returns a REPLY with a KINK_TGT_REP payload containing the TGT.

- When the initiator receives a KINK_TGT_REP, it retrieves a new service ticket with the TGT and retries the command.

This does not directly define a method to detect a dead User-to-User peer, but to recover from the situation that the responder does not have an appropriate TGT to decrypt a service ticket sent from the initiator. After recovery, they can exchange their epochs, and usual DPD mechanism will detect a dead peer if it really has been dead.

The initiator MUST NOT think the peer has been dead on the receipt of a KINK_TGT_REP because of two reasons. One is that the message is not authenticated, and the other is that losing a TGT does not necessarily mean losing the SA database information. The initiator SHOULD NOT forget the previous service ticket until the new one is successfully obtained in order to reduce the cost when a forged KINK_TGT_REP is received.

## 4. KINK Message Format

All values in KINK are formatted in network byte order (most significant byte first). The RESERVED fields MUST be set to zero (0) when a packet is sent. The receiver MUST ignore these fields.

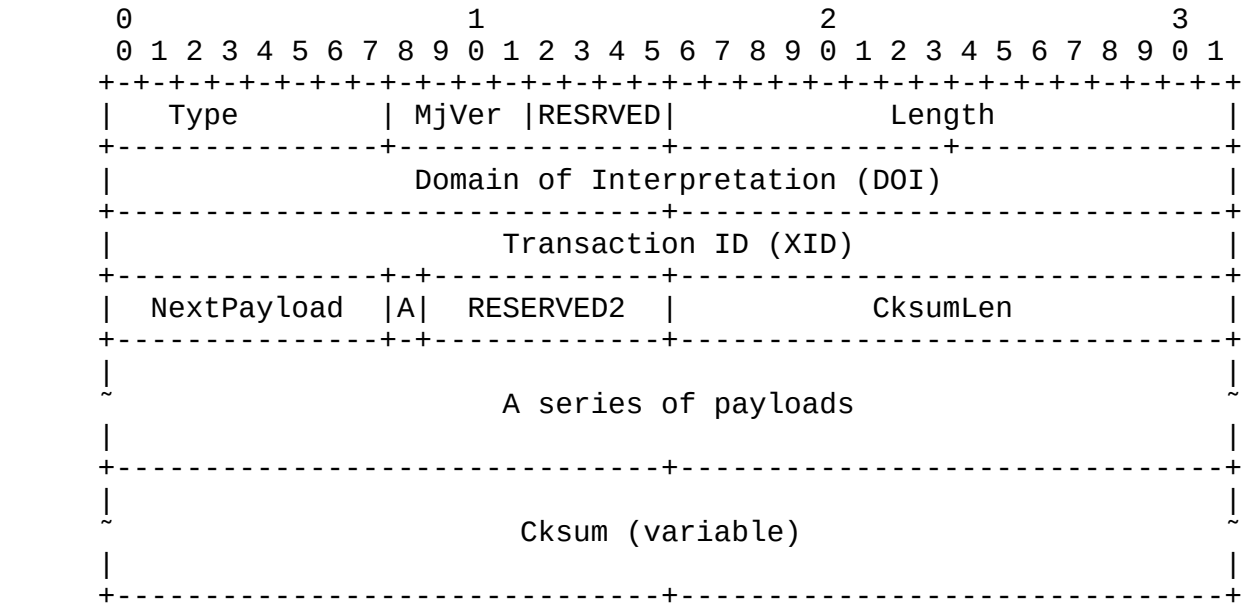


Figure 5: Format of a KINK Message

Fields:

- Type (1 octet) -- The type of this message.

| Type | Value |
|---|---|
| RESERVED | 0 |
| CREATE | 1 |
| DELETE | 2 |
| REPLY | 3 |
| GETTGT | 4 |
| ACK | 5 |
| STATUS | 6 |
| RESERVED TO IANA | 7 - 127 |
| Private Use | 128 - 255 |

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

- RESRVED (4 bits) -- Reserved and MUST be zero when sent, MUST be ignored when received.

- Length (2 octets) -- Length of the message in octets. It is not forbidden in KINK that there are unnecessary data after the message, but the Length field MUST represent the actual length of the message.

- DOI (4 octets) -- The domain of interpretation. All DOIs must be registered with the IANA in the ISAKMP Domain of Interpretation section of the isakmp-registry [ISAKMP-REG]. The IANA Assigned Number for the Internet IP Security DOI [IPDOI] is one (1). This field defines the context of all sub-payloads in this message. If sub-payloads have a DOI field (e.g., 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 acknowledgement. Transaction IDs are used by the initiator to discriminate between multiple outstanding requests to a responder. 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. XIDs MAY be constructed by using a monotonic counter or random number generator.

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

- A, or ACKREQ (1 bit) -- ACK Request. Set to one if the responder requires an explicit acknowledgement that a REPLY was received. An initiator MUST NOT set this flag, nor should a responder except for a REPLY to a CREATE when the optimistic proposal is chosen.

- RESERVED2 (7 bits) -- Reserved and MUST be zero on send, MUST be ignored by a receiver.

- CksumLen (2 octets) -- CksumLen is the length in octets of the cryptographic checksum of the message. A CksumLen of zero implies that the message is unauthenticated.

- Cksum (variable) -- Kerberos keyed checksum over the entire message excluding the Cksum field itself. When any padding bytes are required between the last payload and the Cksum field, they MUST be included in the calculation. 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 content of this field is the output of the Kerberos 5 get_mic function [KCRYPTO]. The get_mic function used is specified by a checksum type, which is a "required checksum mechanism" of the etype for the Kerberos session key in the Kerberos ticket. If the checksum type is not a keyed algorithm, the message MUST be rejected.

  To compute the checksum, the CksumLen field is zeroed out and the Length field is filled with the total packet length without the checksum. Then, the packet is passed to the get_mic function and its output is appended to the packet. Any KINK padding after the Cksum field is not allowed, except the Kerberos internal one, which may be included in the output of the get_mic function. Finally, the CksumLen field is filled with the checksum length and the Length field is filled with the total packet length including the checksum.

  To verify the checksum, a length-without-checksum is calculated from the value of Length field, subtracting the CksumLen. The Length field is filled with the length-without-checksum value and the CksumLen field is zeroed out. Then, the packet without checksum (offset from 0 to length-without-checksum minus 1 of the received packet) and the checksum (offset from length-without-checksum to the last) are passed to the verify_mic function. If verification fails, the message MUST be dropped.

The KINK header is followed immediately by a series of Type/Length/Value fields, defined in section 4.2.

## 4.1. KINK Alignment Rules

KINK has the following rules regarding alignment and padding:

- All length fields MUST reflect the actual number of octets in the structure; i.e., they do not account for padding bytes required by KINK alignments.

- KINK headers, payloads, and the Cksum field MUST be aligned on 4-octet boundaries.

- Variable length fields (except the Cksum field) MUST always start immediately after the last octet of the previous field. That is, they are not aligned to 4-octet boundaries.

## 4.2. 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 aligned to 4-octet boundaries, though the payload length field MUST accurately reflect the actual number of octets in the payload.

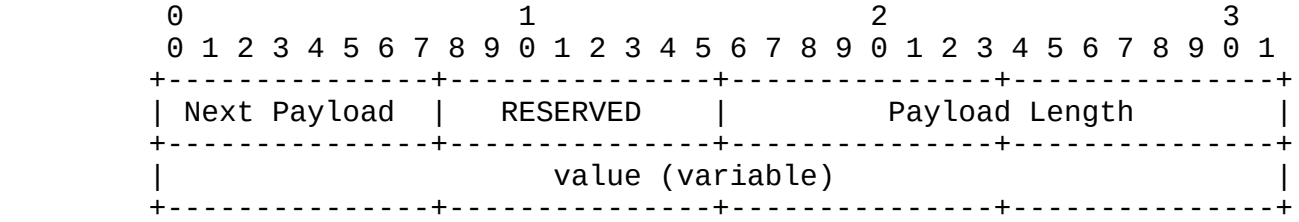


Figure 6: Format of a KINK Payload

Fields:

- Next Payload (1 octet) -- The type of the next payload.

| NextPayload | Value |
|---|---|
| KINK_DONE | 0 |
| KINK_AP_REQ | 1 |
| KINK_AP_REP | 2 |
| KINK_KRB_ERROR | 3 |
| KINK_TGT_REQ | 4 |
| KINK_TGT_REP | 5 |
| KINK_ISAKMP | 6 |
| KINK_ENCRYPT | 7 |
| KINK_ERROR | 8 |
| RESERVED TO IANA | 9 - 127 |
| Private Use | 128 - 255 |

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

- RESERVED (1 octet) -- Reserved and MUST be set to zero by a sender, MUST be ignored by a receiver.

- Payload 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.

### 4.2.1. KINK_AP_REQ Payload

The KINK_AP_REQ payload relays a Kerberos AP-REQ to the responder. The AP-REQ MUST request mutual authentication.

This document does not specify how to generate the principal name. That is, complete principal names may be stored in local policy, Fully Qualified Domain Names (FQDNs) may be converted to principal names, IP addresses may be converted to principal names by secure name services, etc., but see the first paragraph of the Security Considerations section.

If the peer's principal name for the KINK service is generated from an FQDN, the principal name, which the initiator starts from, will be "kink/fqdn@REALM"; where "kink" is a literal string for the KINK IPsec service, "fqdn" is the fully qualified domain name of the service host, and "REALM" is the Kerberos realm of the service. A principal name is case sensitive, and "fqdn" part MUST be lowercase as described in [KERBEROS].

The value field of this payload has the following format:

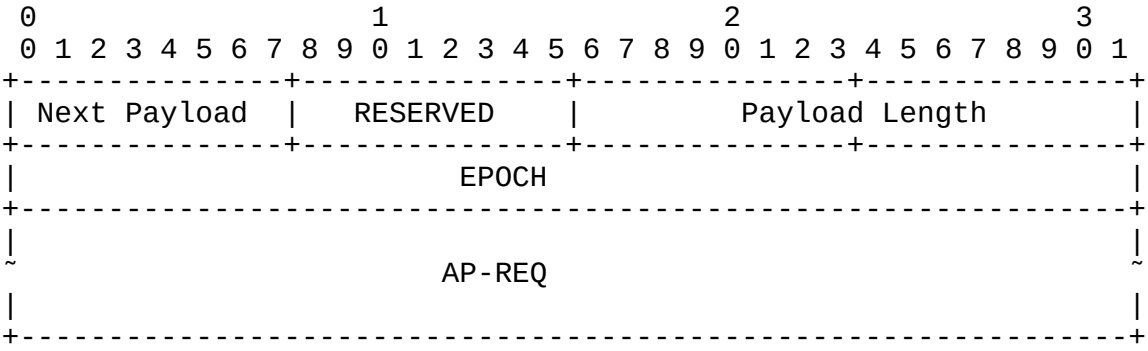


Figure 7: KINK_AP_REQ Payload

Fields:

- Next Payload, RESERVED, Payload Length -- Defined in the beginning of this section.

- EPOCH -- The absolute time at which the creator of the AP-REQ has valid SA information. Typically, this is when the KINK keying daemon started if it does not retain SA information across restarts. The value in this field is the least significant 4 octets of so-called POSIX time, which is the elapsed seconds (but without counting leap seconds) from 1970-01-01T00:00:00 UTC. For example, 2038-01-19T03:14:07 UTC is represented as 0x7fffffff.

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

### 4.2.2. 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:

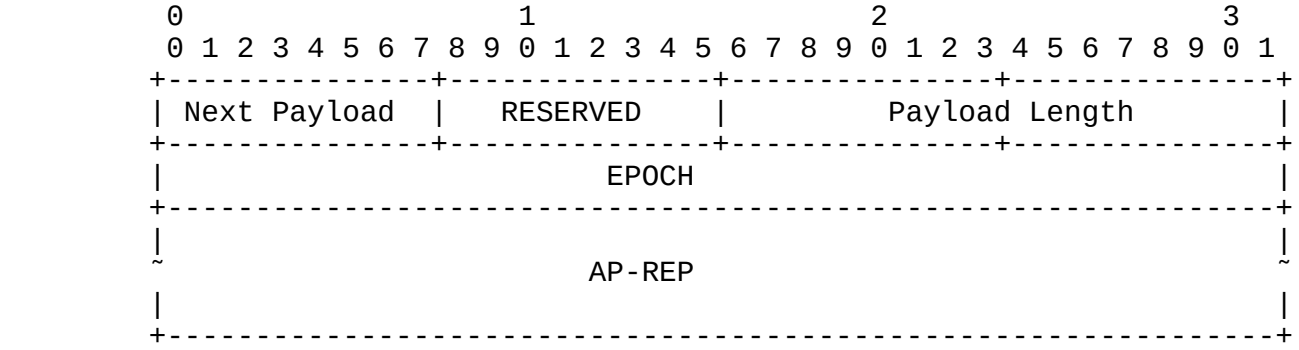


Figure 8: KINK_AP_REP Payload

Fields:

- Next Payload, RESERVED, Payload Length -- Defined in the beginning of this section.

- EPOCH -- The absolute time at which the creator of the AP-REP has valid SA information. Typically, this is when the KINK keying daemon started if it does not retain SA information across restarts. The value in this field is the least significant 4 octets of so-called POSIX time, which is the elapsed seconds (but without counting leap seconds) from 1970-01-01T00:00:00 UTC. For example, 2038-01-19T03:14:07 UTC is represented as 0x7fffffff.

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

#### 4.2.3. KINK_KRB_ERROR Payload

The KINK_KRB_ERROR payload relays Kerberos type errors back to the initiator. The initiator MUST be prepared to receive any valid Kerberos error type [KERBEROS].

KINK implementations SHOULD make use of a KINK Cksum field when returning KINK_KRB_ERROR and the appropriate service key is available. Especially in the case of clock skew errors, protecting the error at the server creates a better user experience because it does not require clocks to be synchronized. However, many Kerberos implementations do not make it easy to obtain the session key in order to protect error packets. For unauthenticated Kerberos errors, the initiator 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:

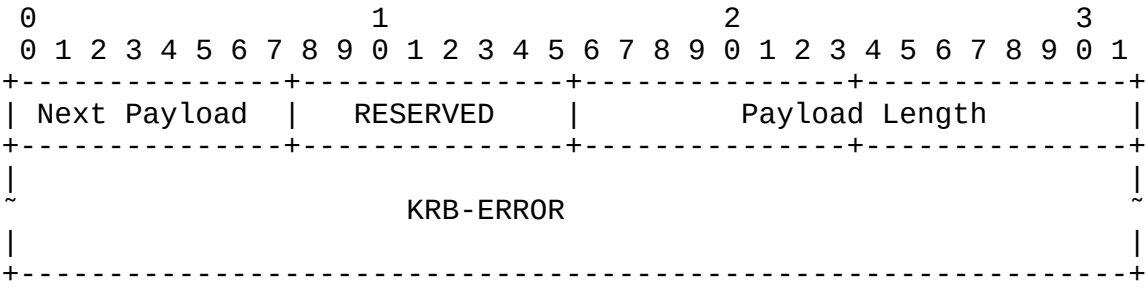


Figure 9: KINK_KRB_ERROR Payload

Fields:

- Next Payload, RESERVED, Payload Length -- Defined in the beginning of this section.
- KRB-ERROR -- The value field of this payload contains a raw Kerberos KRB-ERROR.

### 4.2.4. 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:

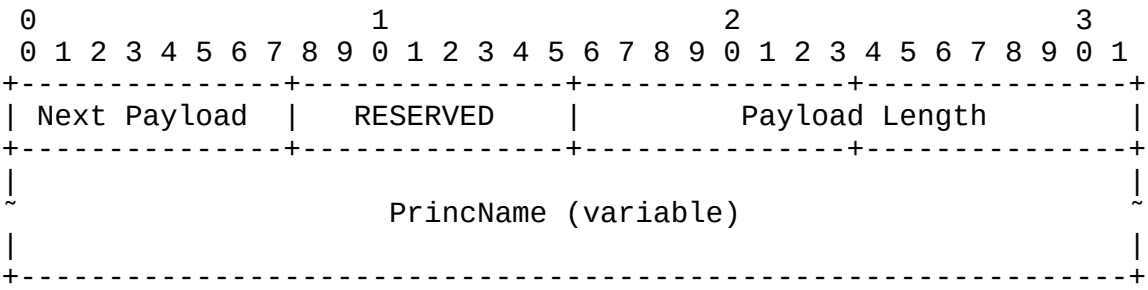


Figure 10: KINK_TGT_REQ Payload

Fields:

- Next Payload, RESERVED, Payload Length -- Defined in the beginning of this section.

- PrincName -- The name of the principal that the initiator wants to communicate with. It is assumed that the initiator knows the responder's principal name (including the realm name) in the same way as the non-User-to-User case. The TGT returned MUST NOT be an inter-realm TGT and its cname and crealm MUST match the requested principal name, so that the initiator can rendezvous with the responder at the responder's realm.

  PrincName values are octet string representations of a principal and realm name formatted just like the octet string used in the "NAME" component of Generic Security Service Application Program Interface (GSS-API) [RFC2743] exported name token for the Kerberos V5 GSS-API mechanism [RFC1964]. See RFC 1964, section 2.1.3.

If the responder is not the requested principal and is unable to get a TGT for the name, it MAY return a KRB_AP_ERR_NOT_US. If the administrative policy prohibits returning a TGT, it MAY return a KINK_U2UDENIED.

### 4.2.5. KINK_TGT_REP Payload

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

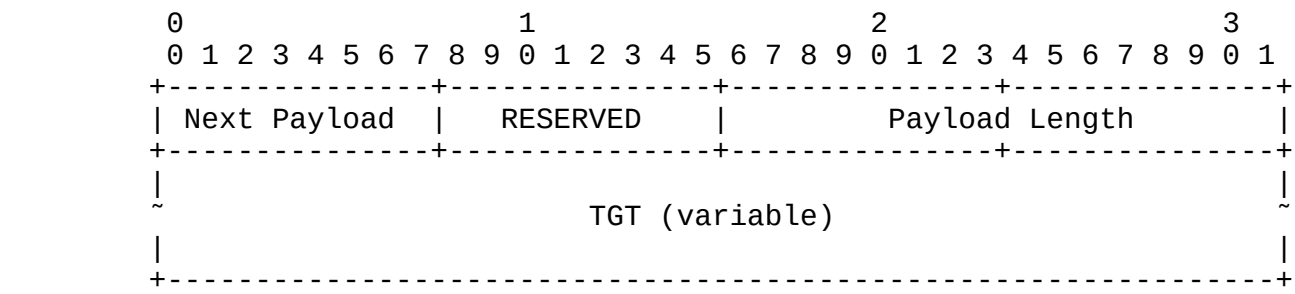


Figure 11: KINK_TGT_REP Payload

Fields:

- Next Payload, RESERVED, Payload Length -- Defined in the beginning of this section.
- TGT -- The Distinguished Encoding Rules (DER)-encoded TGT of the responder.

### 4.2.6. KINK_ISAKMP Payload

The value field of this payload has the following format:

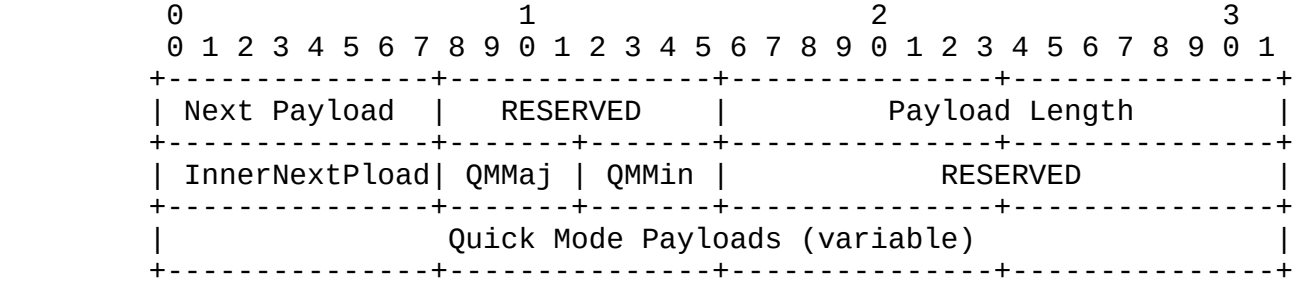


Figure 12: KINK_ISAKMP Payload

Fields:

- Next Payload, RESERVED, Payload Length -- Defined in the beginning of this section.
- 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.

The KINK_ISAKMP payload encapsulates the IKE Quick Mode (phase 2) 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 SAs 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 implementation 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.

### 4.2.7. KINK_ENCRYPT Payload

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

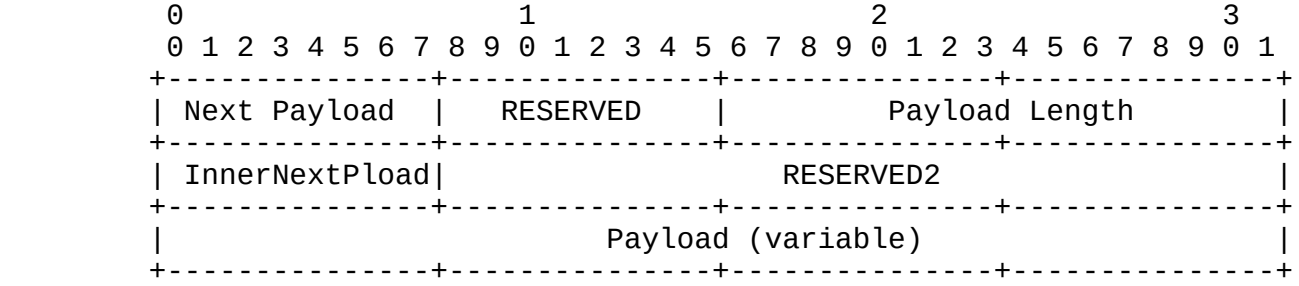


Figure 13: KINK_ENCRYPT Payload

Fields:

- Next Payload, RESERVED, Payload Length -- Defined in the beginning of this section. This payload is the last one in a message, and accordingly, the Next Payload field must be KINK_DONE (0).

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

- RESERVED2 -- Reserved and MUST be zero when sent, MUST be ignored when received.

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

The format of the encryption payload follows the normal Kerberos semantics. Its content is the output of an encrypt function defined in the Encryption Algorithm Profile section of [KCRYPTO]. Parameters such as encrypt function itself, specific-key, and initial state are defined with the etype. The encrypt function may have padding in itself and there may be some garbage data at the end of the decrypted plaintext. A KINK implementation MUST be prepared to ignore such padding after the last sub-payload inside the KINK_ENCRYPT payload. Note that each encrypt function has its own integrity protection mechanism. It is redundant with the checksum in the KINK header, but this is unavoidable because it is not always possible to remove the integrity protection part from the encrypt function.

### 4.2.8. 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 that have their own payloads defined. The error code is in network order.

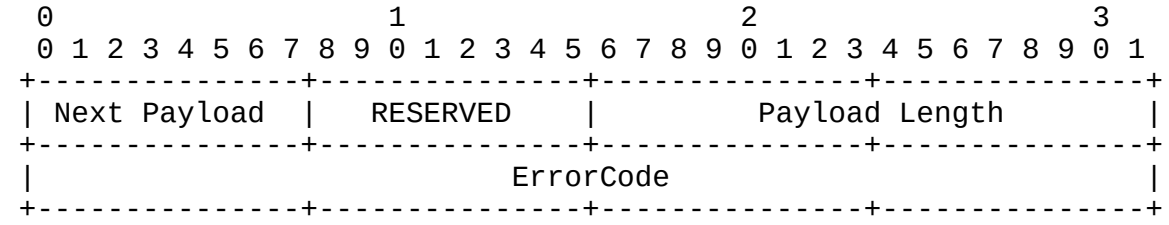


Figure 14: KINK_ERROR Payload

Fields:

- Next Payload, RESERVED, Payload Length -- Defined in the beginning of this section.

- ErrorCode -- One of the following values in the network byte order:

| ErrorCode | Value | 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 |
| RESERVED | 4 | |
| KINK_INTERR | 5 | An unrecoverable internal error |
| KINK_BADQMVERS | 6 | Unsupported Quick Mode Version |
| KINK_U2UDENIED | 7 | Returning a TGT is prohibited |
| RESERVED TO IANA | 8 - 8191 | |
| Private Use | 8192 - 16383 | |
| RESERVED | 16384 - | |

The responder MUST NOT return KINK_OK. When received, the initiator MAY act as if the specific KINK_ERROR payload were not present. If the initiator supports multiple Quick Mode versions or DOIs, KINK_BADQMVERS or KINK_INVDOI is received, and the Cksum is verified, then it MAY retry with another version or DOI. A responder SHOULD return a KINK error with KINK_INVMAJ, when it receives an unsupported KINK version number in the header. When KINK_U2UDENIED is received, the initiator MAY retry with the non-User-to-User mode (if it has not yet been tried).

In general, the responder MAY choose to return these errors in reply to unauthenticated commands, but SHOULD take care to avoid being involved in denial of service attacks. Similarly, the initiator MAY choose to act on unauthenticated errors, but SHOULD take care to avoid denial of service attacks.

## 5. Differences from IKE Quick Mode

KINK directly uses ISAKMP payloads to negotiate SAs. In particular, KINK uses IKE phase 2 payload types (aka Quick Mode). In general, there should be very few changes necessary to an IKE implementation to establish the SAs, and unless there is a note to the contrary in the memo, all capabilities and requirements in [IKE] MUST be supported. IKE phase 1 payloads MUST NOT be sent.

Unlike IKE, KINK defines specific commands for creation, deletion, and status of SAs, mainly to facilitate predictable SA creation/deletion (see sections 3.2 and 3.3). 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 2 payload is not mentioned here, it means that there are no differences in its use.

- The Security Association Payload header for IP is defined in section 4.6.1 of [IPDOI]. 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 that it defines an optimistic 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. For this purpose, KINK MUST use a pseudo-random function determined by the etype of the session key.

- KINK does not use the HASH payload at all.

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

### 5.1. Security Association Payloads

KINK supports the following SA 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 of these attributes.

### 5.2. 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.

### 5.3. Identification Payloads

The Identification payload carries information that is used to identify the traffic that is to be protected by the SA that will be established. KINK restricts the ID types, which are defined in section 4.6.2.1 of [IPDOI], 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 |

### 5.4. Nonce Payloads

The Nonce payload contains random data that MUST be used in key generation. It MUST be sent by the initiating KINK peer, and MAY be sent by the responding KINK peer. See section 7 for the discussion of its use in key generation.

## 5.5. Notify Payloads

Notify payloads are used to transmit several informational data, such as error conditions and state transitions to a peer. For example, notification information transmit 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.

Types in the range 0 - 16383 are intended for reporting errors [ISAKMP]. An implementation receiving a type in this range that it does not recognize in a response MUST assume that the corresponding request has failed entirely. Unrecognized error types in a request and status types in a request or response MUST be ignored, and they SHOULD be logged. Notify payloads with status types MAY be added to any message and MUST be ignored if not recognized. They are intended to indicate capabilities, and as part of SA negotiation are used to negotiate non-cryptographic parameters.

The table below lists the Notification messages and their corresponding values. PAYLOAD-MALFORMED denotes some error types defined by [ISAKMP]. Hence INVALID-PROTOCOL-ID, for example, is not used in this document. INVALID-MAJOR-VERSION and INVALID-MINOR-VERSION are not used because KINK_BADQMVERS is used to tell the initiator that the version of IKE is not supported.

| NOTIFY MESSAGES - ERROR TYPES | Value |
|---|---|
| INVALID-PAYLOAD-TYPE | 1 |

Sent if the ISAKMP payload type is not recognized. It is also sent when the KE payload is not supported by the responder. Notification Data MUST contains the one-octet payload type.

| | |
|---|---|
| INVALID-SPI | 11 |

Sent if the responder has an SPI indicated by the initiator in case of CREATE flow, or if the responder does not have an SPI indicated by the initiator in case of DELETE flow.

| | |
|---|---|
| NO-PROPOSAL-CHOSEN | 14 |

Sent if none of the proposals in the SA payload was acceptable.

PAYLOAD-MALFORMED 16

Sent if the KINK_ISAKMP payload received was invalid because some type, length, or value was out of range. It is also sent when the request was rejected for reason that was not matched with other error types.

### 5.6. Delete Payloads

KINK directly uses ISAKMP Delete payloads with no changes.

### 5.7. KE Payloads

IKE requires that perfect forward secrecy (PFS) be supported through the use of the KE payload. KINK retains the ability to use PFS, but relaxes the requirement from must implement to SHOULD implement. The reasons are described in the Security Considerations section.

## 6. Message Construction and Constraints for IPsec DOI

All commands, responses, and acknowledgements 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 the Kerberos mechanisms. In addition, commands and responses MUST use a cryptographic checksum over the entire message if the two peers share a key via a ticket exchange.

In all cases in this section, if a message contains a KINK_AP_REQ or KINK_AP_REP payload, other KINK payloads MAY be encapsulated in a KINK_ENCRYPT payload.

### 6.1. REPLY Message

The REPLY message is a generic reply that MUST contain either a KINK_AP_REP, a KINK_KRB_ERROR, or a KINK_ERROR payload. REPLY messages MAY contain additional DOI-specific payloads such as ISAKMP payloads that are defined in the following sections.

### 6.2. ACK Message

ACKs are sent only when the ACKREQ bit is set in a REPLY message. An ACK message MUST contain an AP-REQ payload and no other payload.

### 6.3. CREATE Message

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 payloads
         SA Payload
              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 payloads
         SA Payload
              Proposal Payloads
                   Transform Payload
         [Nonce Payload (Nr)]
         [KE]
         [IDci, IDcr]
         [Notification Payloads]
```

Note that there MUST be at least a single proposal payload and a single transform payload in REPLY messages. There will be multiple proposal payloads only when an SA bundle is negotiated. Also: unlike IKE, the Nonce payload Nr is not required, and if it exists, an acknowledgement must be requested to indicate that the initiator's outgoing SAs must be modified. If any of the first proposals are not chosen by the recipient, it SHOULD include the Nonce payload.

KINK, like IKE, allows the creation of many SAs in one create command. If any of the optimistic proposals are not chosen by the responder, it MUST request an ACK.

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

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

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

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

### 6.4. DELETE Message

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_ISAKMP payloads
         Delete Payloads
         [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]
     KINK_ISAKMP payloads
         Delete Payloads
         [Notification Payloads]
```

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

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

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

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

### 6.5. STATUS Message

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
[[KINK_ENCRYPT]
   KINK_ISAKMP payload
       [Notification Payloads]]
```

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

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

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

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

### 6.6. GETTGT Message

A GETTGT command is only used to carry a Kerberos TGT and is not related to SA management; therefore, it contains only KINK_TGT_REQ payload and does not contain any DOI-specific payload.

There are two forms of replies for a GETTGT. In the normal form, where the responder is allowed to return its TGT, the REPLY contains KINK_TGT_REP payload. If the responder is not allowed to return its TGT, it MUST reply with a KINK_ERROR payload.

## 7. ISAKMP Key Derivation

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

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

The following differences apply:

- prf is the pseudo-random function corresponding to the session key's etype. They are defined in [KCRYPTO].
- SKEYID_d is the session key in the Kerberos service ticket from the AP-REQ. Note that subkeys are not used in KINK and MUST be ignored if received.
- Both Ni_b and Nr_b are the part of the Nonce payloads (Ni and Nr, respectively) as described in section 3.2 of [IKE]. Nr_b is optional, which means that Nr_b is treated as if a zero length value was supplied when the responder's nonce (Nr) does not exist. When Nr exists, Nr_b MUST be included in the calculation.

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].

The rest of the key derivation (e.g., how to expand KEYMAT) follows IKE. How to use derived keying materials is up to each service (e.g., section 4.5.2 of [IPSEC]).

## 8. Key Usage Numbers for Kerberos Key Derivation

Kerberos encrypt/decrypt functions and get_mic/verify_mic functions require "key usage numbers". They are used to generate specific keys for cryptographic operations so that different keys are used for different purposes/objects. KINK uses two usage numbers, listed below.

| Purpose | Usage number |
|---|---|
| KINK_ENCRYPT payload (for encryption) | 39 |
| Cksum field (for checksum) | 40 |

## 9. Transport Considerations

KINK uses UDP on port 910 to transport its messages. There is one timer T which SHOULD take into consideration round-trip considerations and MUST implement a truncated exponential back-off mechanism. The state machine is simple: any message that 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 re-created 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 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 that are outstanding for any particular transaction.

When a KINK peer retransmits a command, it MUST use the same ticket within the retransmissions. This is to avoid race conditions on using different keys, which result in different KEYMATs between an initiator and a responder. For this reason, (1) an initiator MUST obtain a ticket whose lifetime is greater than the initiator's maximum transaction time including timeouts, or (2) it MUST continue to use the same ticket within a set of retransmissions, and iff it receives an error (most likely KRB_AP_ERR_TKT_EXPIRED) from the responder, it starts a new transaction with a new ticket.

## 10. Security Considerations

The principal names are the identities of the KINK services, but the traffic protected by SAs are identified by DOI-specific selectors (IP addresses, port numbers, etc.). This may lead to a breakaway of SA-protected data from authentication. For example, if two different hosts claim that they have the same IP address, it may be impossible to predict which principal's key protects the data. Thus, an implementation must take care for the binding between principal names and the SA selectors.

Sending errors without cryptographic protection must be handled very carefully. There is a trade-off between wanting to be helpful in diagnosing a problem and wanting to avoid being a dupe in a denial of service attack.

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 2 payloads to create and delete SAs; the security considerations which pertain to IKE phase 1 may be safely ignored. However, being able to ignore IKE's authentication phase necessarily means that KINK inherits all of the security considerations of Kerberos authentication as outlined in [KERBEROS]. For one, a KDC, like an Authentication, Authorization, and Accounting (AAA) server, is a point of attack and all that implies. Much has been written about various shortcomings and mitigations of Kerberos, and they should be evaluated for any deployment.

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 should 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 does not currently provide perfect forward secrecy in general. KINK with the KE payload can provide PFS for a service key from a Kerberos key, but the KE is not mandatory because of the computational cost. This is a trade-off and operators can choose the PFS over the cost, and vice versa. KINK itself should be secure from offline analysis from compromised principal passphrases if PFS is used, but from an overall system's standpoint, the existence of other Kerberized services that do not provide PFS makes this a less than optimal situation.

## 11. IANA Considerations

The IANA has assigned a well-known port number for KINK.

The IANA has created a new registry for KINK parameters, and has registered the following identifiers.

KINK Message Types (section 4)
KINK Next Payload Types (section 4.2)
KINK Error Codes (section 4.2.8)

Changes and additions to this registry follow the policies described below. Their meanings are described in [BCP26].

- Using the numbers in the "Private Use" range is Private Use.
- Assignment from the "RESERVED TO IANA" range needs Standards Action, or non-standards-track RFCs with Expert Review. (Though the full specification may be a public and permanent document of a standards body other than IETF, an RFC referring it is needed.)
- Other change requires Standards Action.

## 12. Forward Compatibility Considerations

KINK can accommodate future versions of Quick Mode through the use of the version field in the ISAKMP payload as well as new domains of interpretation. In this memo, the only supported Quick Mode version is 1.0, which corresponds to [IKE]. Likewise, the only DOI supported is the IPsec domain of interpretation [IPDOI]. New Quick Mode versions and DOIs MUST be described in subsequent memos.

KINK implementations MUST reject ISAKMP versions that are greater than the highest currently supported version with a KINK_BADQMVERS error type. A KINK implementation that receives a KINK_BADQMVERS message SHOULD be capable of reverting back to version 1.0.

### 12.1. New Versions of Quick Mode

The IPsec working group is defining the next-generation IKE protocol [IKEv2], which does not use Quick Mode, but it is similar to the one in IKEv1. The difference between the two is summarized in Appendix A of [IKEv2]. Each of them must be considered in order to use IKEv2 with KINK.

### 12.2. New DOI

The KINK message header contains a field called "Domain of Interpretation (DOI)" to allow other domains of interpretation to use KINK as a secure transport mechanism for keying.

As one example of a new DOI, the MSEC working group defined the Group Domain of Interpretation [GDOI], which defines a few new messages, which look like ISAKMP messages, but are not defined in ISAKMP.

In order to carry GDOI messages in KINK, the DOI field in the KINK header would indicate that GDOI is being used, instead of IPSEC-DOI, and the KINK_ISAKMP payload would contain the payloads defined in the GDOI document rather than the payloads used by [IKE] Quick Mode. The version number in the KINK_ISAKMP header is related to the DOI in the KINK header, so a maj.min version 1.0 under DOI GDOI is different from a maj.min version 1.0 under DOI IPSEC-DOI.

## 13. Related Work

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

- a mechanism for encrypting and authenticating IP datagram payloads that 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 a pre-shared key, key distribution is required between all peers -- an O(n^2) problem -- which is problematic for large deployments.

## 14. Acknowledgements

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 simplified version of Kerberized IPsec, including Sasha Medvinsky, Mike Froh, and Matt Hur and David McGrew. The inspiration for wholly reusing IKE phase 2 is the result of Tero Kivinen's document suggesting grafting Kerberos authentication onto Quick Mode.

## 15. References

### 15.1. Normative References

[BCP26] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 2434, October 1998.

[IKE] Harkins, D. and D. Carrel, "The Internet Key Exchange (IKE)", RFC 2409, November 1998.

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

[IPSEC] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, December 2005.

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

[ISAKMP-REG] IANA, "Internet Security Association and Key Management Protocol (ISAKMP) Identifiers", <http://www.iana.org/assignments/isakmp-registry>.

[KCRYPTO] Raeburn, K., "Encryption and Checksum Specifications for Kerberos 5", RFC 3961, February 2005.

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

[RFC1964] Linn, J., "The Kerberos Version 5 GSS-API Mechanism", RFC 1964, June 1996.

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

## 15.2. Informative References

[GDOI] Baugher, M., Weis, B., Hardjono, T., and H. Harney, "The Group Domain of Interpretation", RFC 3547, July 2003.

[IKEv2] Kaufman, C., "Internet Key Exchange (IKEv2) Protocol", RFC 4306, December 2005.

[PKINIT] Zhu, L. and B. Tung, "Public Key Cryptography for Initial Authentication in Kerberos", Work in Progress, February 2006.

[REQ4KINK] Thomas, M., "Requirements for Kerberized Internet Negotiation of Keys", RFC 3129, June 2001.

[RFC793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[RFC2743] Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, January 2000.

Authors' Addresses

Shoichi Sakane
Yokogawa Electric Corporation
2-9-32 Nakacho, Musashino-shi,
Tokyo 180-8750 Japan

EMail: Shouichi.Sakane@jp.yokogawa.com

Ken'ichi Kamada
Yokogawa Electric Corporation
2-9-32 Nakacho, Musashino-shi,
Tokyo 180-8750 Japan

EMail: Ken-ichi.Kamada@jp.yokogawa.com

Michael Thomas
Cisco Systems
170 West Tasman Drive
San Jose, CA 95134

EMail: mat@cisco.com

Jan Vilhuber
Cisco Systems
170 West Tasman Drive
San Jose, CA 95134

EMail: vilhuber@cisco.com

## Full Copyright Statement



## Intellectual Property



## Acknowledgement

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