A distributed fault-tolerant group key selection protocol for MACsec

Mick Seaman

This note describes a Key Selection Protocol (KSP) that allows MACsec participants to prove mutual authentication through common possession of a master key, agree on SAKs, and ensure that data traffic and keys are neither being delayed nor replayed.

This second revision changes the way that KSP messages are themselves protected. It also allows for key selection and distribution by an elected group leader—as an alternative to independent key derivation by each participant using the public key contributions and the secret master key. The key contributions themselves are retained when key distribution is used, as they provide a reliable indicator from each participant that a fresh key is required, and a reliable indicator to each participant that a fresh key has indeed been provided.

1. Introduction

KSP is designed to meet the need for the participants in a IEEE 802.1AE MACsec secure connectivity association (CA) to agree data keys (SAKs) for symmetric shared key cryptography. [1]

KSP assumes an attacker can copy any frame sent by any participant, can selectively prevent delivery to some or all participants, can transmit arbitrary frames to arbitrary participants[2], and can control physical connectivity and power to each participant. KSP maintains secure connectivity against attacks, using some or all of these mechanisms, short of a “wire cutting attack” that simply denies service. KSP also supports procedures designed to maintain security after theft of equipment containing keys.

KSP’s requirements are driven by MACsec’s role in underpinning basic network connectivity in enterprise and provider bridged network (802.1ad) environments. MACsec secures individual LANs, or virtual LANs over a bridged network, protecting the infrastructure as well as access to that infrastructure[3]. Connectivity may need to be maintained for a number of years, with individual participants in the CA being powered up or down or being changed during that time. Key selection during times of network change can depend neither on the presence of a previously selected distinguished participant nor on connectivity from that or any other participant to an authentication server. However it is highly desirable that authentication and authorization be manageable under a AAA framework. Each KSP Entity starts its operation with a pre-shared master key known as the CAK (connectivity association key), whose mutual possession authenticates participants in the CA. The CAK will often be a PMK distributed by 802.1X/EAP[4,5].

KSP uses the CAK to agree a sequence of SAKs, while ensuring that an SAK with a PN (packet number) combination is never repeated, either accidentally or as a result of a deliberate attack, as it serves as a cryptographic nonce in MACsec. KSP manages the use of overlapping SAKs to provide continuous connectivity when they are being changed. Connectivity is also maintained through changes of CAK.

A CA may change from being logically point-to-point (with only two participants or members) to

[1] The initial KSP proposal (4/7/2004) used the MI.MN tuples as described in this note to prove liveness and protect against delay and replay, but supported central distribution of a time-bounded key by an elected (or recognized) participant. Following discussion this was changed (10/04) to use a key contribution methodology. The development of KSP has benefited from discussions with John Viega, David McGrew, and Brian Weis though this does not necessarily signify their approval for any particular part of the design. All errors and omissions are my own.

[2] The combination of these possibilities covers misordering of messages, which can also happen quite unintentionally.

[3] MACsec does not attempt to supplant IPsec by providing end to end security.

[4] Each participant can simultaneously participate in more than one instance of KSP, each distinguished by a separate CAK. This is important because authentication protocol (EAP in particular) does not provide mutual confirmation of the delivery of PMK to the mutually authenticated parties, or procedures to support symmetrical networking scenarios with delivery of a single PMK, or mechanisms robust against timer choices that reliably deliver a single PMK in the face of packet loss, delay, and transmission retry in the network supporting the EAP exchange. The way to check that all parties have a PMK is to attempt to use it, and to pick between two PMKs that any participant may find equally attractive is to try both and select one that works.

[5] KSP can also be used with a PMK as its CAK for pair-wise delivery of a (the same) group CAK to each CA member. This aspect of its operation needs to be explained further, but requires only minor additions to the key distribution additions introduce in this revision. It provides a simple way for a Supplicant to either participate in a point-to-point CA or to discover that it is to join a group CA, and allows an Authenticator to add a Supplicant to a group CA, and to furnish the members of the group CA with fresh CAKs.

a group CA (three or more members) without disrupting connectivity (for at least some CA members). Group CAs support multicast transmission under a single key to all group members. KSP selects common keys for all the group members, to avoid receivers having separate key tables for every transmitter.

KSP uses a single message type. KSPDUs (KSP protocol data units) are sent periodically by each participant, and as needed subject to rate limiting controls. Each KSPDU is sent as an Ethernet frame with a well known multicast MAC address as its destination address, and supports mutual discovery of participants attached to the same LAN as well as conveying key information. Each KSPDU is integrity protected with the secret master key (CAK), so only stations possessing that master key can generate new KSPDUs and tampering by attackers can be detected. Each protocol participant includes a Member Identifier (MI) and a Message Number (MN), unique for messages transmitted using that identifier, in each transmitted KSPDU. These MI.MN tuples are used to prove liveness, protecting against replay or delay of KSPDUs.

Using this secure transport, each participant transmits a Key Contribution (KC). The KCs of all the active (as proved by the transport) participants are combined under a pseudo-random function using the secret CAK to generate an SAK. The SAK is identified by a Key Identifier (KI) that is simply the exclusive-or of all KCs contributing to the SAK. The status of each participant relative to each SAK— its ability to receive and or transmit using the key, the binding of the key to a MACsec association number (AN), and the lowest PN within an acceptable delay window — are transmitted with reference to the KI.

This revision of KSP also provides for a single participant to select and distribute an SAK to all the other participants. At the time of writing it is not clear whether this should be a dynamic option, i.e. selectable during the course of operation of a single instance of KSP, a static option i.e. one that requires preconfiguration of all the participants, or a protocol design decision. Input to KSP development has variously suggested that it is essential that each participant contribute to the key, and on the other hand that it is essential that the key be chosen by one participant so that it can be distributed by already approved distribution methods. For the present it is sufficient to note that a number of protocol mechanisms underpin both approaches. If the key contribution method were to be removed a different KI format might be chosen and the KCs (which act as requests for a new key) might be shorter, but the KIs and KCs or their renamed equivalents are still required.

When an SAK is chosen and distributed by a single participant, the Member Identifiers are used to choose that participant. A new SAK is only advertized by a participant (the CA Leader) that believes it possesses the highest priority MI[6] of all those proven to be live. Just as for the key contribution method (see below) transmitters only start to use the new key when all currently live receivers have installed it. Each participant's Key Contribution serves as a prompt to request a new key[7]. The CA Leader distributes a fresh SAK whenever a participant joins the CA, leaves the CA, or changes its KC. The KI of the distributed key is part of the KSPDU TLV that delivers the wrapped up key.

2. KSP Overview

This section provides an overview of:

- the secure transport, including
  - addressing and protocol identification
  - integrity protection of KSPDUs
  - proof of master key possession and timeliness
  - message ordering
- use of the secure transport, including
  - key (SAK) agreement and calculation
  - SAK installation, use, and replacement
  - data delay protection

Figure 1 summarizes the format and use of each of the fields of a KSPDU[8]. Figure 2 is an object diagram of a KSP Entity, i.e. that part of a protocol participant that actually implements KSP.

NOTE—Figure 1 does not currently show the key distribution TLV, which is an optional elements preceding the null terminator and ICV. This TLV comprises a key identifier and the SAK, with the latter protected by the AES Key Wrap.

One of the things that the object diagram is good at depicting is the scope and containment of the identifiers and other parameters used by KSP. Whenever an identifier is described as 'const' within its immediately containing object, it can only be changed by destruction of that object and all the others that it contains followed by creation of a new object. Following sections detail the use, creation, destruction, and scope of each of the protocol parameters and objects.

Figure 10 is a state machine, drawn using the conventions commonly used by 802.1[9], that describes the life cycle of an SAK from the point of view of an individual KSP participant. Figure 8 is the state machine that generates fresh SAKs. The object diagram of Figure 2 is realized by

[6] Numerically the highest, with the MIs being interpreted as numbers in network byte order.

[7] An alternative, which may be preferable if there is no key contribution

[8] Since the last published KSP revision the integrity protection of KSPDUs has been changed, and the IV field previously included in the PDU has been removed.

[9] Derived from those used by 802.3.

C++ code and an accompanying English language specification that realize the state machines and other processing requirements of the protocol. Taken together these comprise the definitive specification of KSP.

2.1. Terminology

Active — used to describe a Peer that has proved current possession of the CAK to the Actor . A synonym for Live.

Actor — the KSP participant being discussed or undertaking the action described.

CA — secure Connectivity Association, a MACSec term meaning the subset of stations attached to a LAN that are mutually authenticated and authorized, and use MACsec to exchange data to the exclusion of unauthorized stations. Also used to describe the symmetric and transitive connectivity provided between the stations.

CAK — a master key, distributed to the potential members of the CA prior to the operation of KSP by 802.1X/EAP or other means, possession of the serves to mutually authenticate the CA members. Different instances of KSP, with different CAKs, can be simultaneously active.

CKI — an identifier for the CAK used to protect a particular KSPDU.

DA — destination MAC address.

EUI — an Extended Unique Identifier, a value derived from an OUI (Organizationally Unique Identifier) allocated by the IEEE Registration Authority. Historically an OUI was a MAC Address block and an EUI-48 was a 48-bit MAC Address.

ICV — Integrity Check Value.

initialize, initialized — returning the KSP entity to its initial or power on state. In this state the entity only knows the CAK, CKI, the SCI and any MAC Addresses to be used.

KC — Key Contribution. A random nonce (128 bits) independently chosen by each KSP participants as input to the pseudo-random function of the CAK used to calculate each SAK. Also used to drive the distribution of a fresh SAK when that is chosen by the CA Leader.

KI — Key Identifier. The exclusive-or of the Key Contributions of an actor's Active Peers.

Leader — the active CA member with the highest priority .

Live — a synonym for Active.

LKI — Latest (or proposed) Key Identifier.

LLPN — The LPN for the key corresponding to the LKI.

LPN — Lowest acceptable Packet Number, a field in a KSPDU for each of the possible keys that reflects the lowest PN used in a MACsec data frame protected by he key and transmitted using the actor's SC.

MAC — Media Access Control. An abbreviation used throughout the LAN industry and in most IEEE 802 standards. The term Integrity Check Value (ICV) is used in MACsec for an unrelated security concept that others associate with the acronym MAC.

Member Identifier (MI) — a nonce[10] chosen by the actor to identify itself in subsequent protocol exchanges.

message — synonymous with KSPDU.

Message Number (MN) — a number starting at 1 and incrementing to $2^{32} - 1$ that serves to uniquely identify and order each KSPDU within the context of a Member Identifier. An MI.MN tuple is a nonce for the KSPDU.

OKI — Old Key Identifier.

OLPN — The LPN for the key corresponding to the OKI.

Peer(s) — (an)other KSP participant(s) attached to the same LAN as the actor.

PN — Packet Number. In each MACsec frame the PN is a nonce, i.e. is only used once for the SAK.

SA — Secure Association, a term in general use for the shared information that enables secure communication between entities, but used in this note in the particular sense that MACsec uses it, i.e. the information and relationship between entities that supports MACsec data transfer with a single key. Also the acronym for a source MAC address.

SAK — Secure Association Key, a MACsec term for the key used by one of the SAs that compose an SC.

SC — Secure Channel, a MACSec term meaning the sequence of secure data frames transmitted by a MACsec participant to the other members of the CA.

SCI — Secure Channel Identifier, a MACsec term meaning an EUI-48+16 or EUI-64 identifier for an SC that can form part of the MACsec data frame.

2.2. Addressing

Each KSPDU is transmitted using the MAC Internal Sublayer Service (ISS)[11]. The destination MAC address (DA) is a multicast address from the set of reserved addresses that are filtered by standard bridges. Not only does this correctly bound the discovery of other potential KSP participants to those attached to the same LAN or virtual LAN, but also means

[10] Frankly I get confused by the latitude allowed to the term nonce in security documentation. The Member Identifier is clearly used for a period of time. The only sense in which it is 'used once' is that when it is chosen again any previously used value is vanishingly unlikely to be reselected.

[11] The ISS (Internal Sublayer Service) specified in IEE Std 802.1D Clause 6 is supported by all 802 LAN types. For those not interested in the level of detail and flexibility the ISS provides each KSPDU can be simply regarded as an Ethernet frame.

that an attacker cannot launch an attach from anywhere in a bridged network but has to attach directly to the LAN to be attacked, or at least subvert a station attached to that LAN.

On physical, as opposed to virtual, LAN media the destination MAC Address used with each KSPDU is the Bridge Group Address specified in IEEE Std 802.1D.

The source address (SA) is that of the transmitting station.

The Ethertype is allocated to identify KSP.

Each frame's DA and SA are included in the KSPDU integrity check (see below).

2.3. KSPDU integrity protection

Integrity protection of each KSPDU is accomplished as follows. The octets of the destination and source MAC addresses, in canonical format order, are concatenated with the KSPDU proper, beginning with the Ethertype allocated to identify the protocol, and the resulting octet sequence (M) is protected with CMAC using AES-128 and a subkey derived from the CAK[12]. Specifically a 128 bit Integrity Check Value (ICV) is added to the KSPDU:

ICV = AES-CMAC(K,M,128)

where K is derived from the CAK as follows:

K = AES-ECB(CAK,0x1)

where the CAK is the AES key and the encrypted data is a single 128-bit block with the value '0x1'[13].

The CAK is identified by the CA Key Identifier (CKI), a 64-bit value that is either an EUI-48 (such as a MAC Address) plus a further 16 bits allocated by the entity identified by the EUI-48, or is an EUI-64[14].

Apart from an SAK that is separately encrypted within the optional key distribution TLV[15],none of the information in the KSPDU needs to be kept confidential. This fact is used to facilitate observation and, if necessary, debugging of protocol behavior by operational personnel without having to provide them with the CAK. Most importantly this means that a potentially vulnerable interface or set of procedures do not have to be provided to selectively disclose the CAK for first level network maintenance. Additionally the Key Identifier (KI, see below) conveyed in KSPDUs serves to uniquely identify, to a high probability, the key actually agreed by KSP participants without giving the attacker any information about that key. The KI together with other information in KSPDUs can be communicated insecurely to someone who has knowledge of the CAK if it proves really necessary to determine the actual agreed value of the SAK.

2.4. Proving liveness

Each KSP participant identifies itself within the protocol by a random[16] number, its Member Identifier (MI). Each transmitted KSPDU includes the actor's Member Identifier and a Message Number (MN). MN is initialized to 1 and is incremented in each KSPDU sent. The actor records the last MN used at regular intervals so that it knows the range of MNs it has used within the last one or two seconds.

The Member Identifier is randomly chosen from a sufficiently large number space[17] that it is vanishingly unlikely to have been previously used in combination with the same CAK. Each KSPDU not only contains the actor's MI and the highest value of MN ever used (up to the time of transmission) with that MI, but also lists of MI.MN tuples for each of the other participants. These provide proof of current possession of the CAK to the actor's peers as follows. If a peer P (say) has transmitted $MI_P.MN_P$ within the last second and subsequently receives that tuple in a message from the participant A identified by MI_A with message number MN_A, and integrity protected by the CAK, then P knows that A[18] possesses the CAK and has transmitted that message (and any others received with MI_A and $MN > MN_A$) within the last second.

A peer that has proved current possession of the CAK to the actor is referred to as "live" or "active", and the proof itself as "proof of liveness". A peer that has not proved current possession of the CAK is referred to as a "potential" peer. The MI.MN tuples for an actor's peers is correspondingly organized into two lists in transmitted KSPDUs : a "Live List" and a "Potential List". This allows a participant that has just been attached to a LAN or re-initialized to acquire all the data it needs to prove liveness from by receiving a single KSPDU and including

[12] A suggested by Brian Weis. See Brian's LKS specification and M. Dworkin, "Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication", NIST Special Publication 800-38B, May 2005

[13] An explanation of why it is necessary to ECB the CAK before using it in CMAC would be useful. I think it is because the CAK has also been used in the AES Key Wrap, and the ECB is being used to derived subkeys from the CAK rather than relyiing on any proof that two different direct uses of the CAK can never cause an exposure.

[14] There are alternatives, as the CKI only needs to have a very good chance of uniqueness. Collisions cause failures to communicate, not breaches of security. A hash of a key name could be used. 64 bits was chosen previously to align the initial PDU format with that for MACsec when GMAC protection of KSPDUs was advocated. This is no longer a consideration.

[15] Used when SAKs are chosen and distributed by a CA Leader, rather than being computed from the KCs by each participant.

[16] A number chosen so that its distribution across all KSP participants is indistinguishable from random.

[17] 96 bits in the current specification, though this could be easily increased if that is thought too small.

[18] Strictly speaking P knows that some station in possession of the CAK has sent the message with $MI_A.MN_A$. Like all protocols this cannot work if authorized participants don't follow the protocol but send bogus information. The essential point is that an attacker can't use old replayed messages to inject out of date information.

all the tuples in the "Live List" in its own "Potential List". Separation of the two lists avoids the members of the CA keeping alive the memory of a long departed participant.

The Member Identifier for a participant is chosen afresh whenever the Message Number space is exhausted, or when a "collision" is detected, i.e. a participant receives a PDU that is did not transmit with its own MI as the transmitter's MI. Both of these are unlikely events, as the MN space should last for over 10 years even at the transmission limiting rate of 10 KSPDUs per second — far in excess of any reasonable policy for changing the CAK, which creates a new KSP instance with a fresh MI — while the change of an MI collision even in a very large CA of a 100 members is less than 1 in 2^{90}. However the mechanisms for change are included to avoid the proof of protocol correctness depending on can't happen events — it is an axiom of sound protocol design that convergence to a known good state from any state should happen within known bounded time given correct protocol execution during that time without regard to prior history and arguments that such and such a bad state is unreachable ("can't happen").

2.5. Message ordering

Since the Message Number (MN) increments with each transmission for a participant identified by a given Member Identifier, KSP allows each authorized participant to convey an ordered sequence of messages to other participants attached to the same LAN and possessing the same CAK, while allowing delayed or replayed messages to be discarded.

This allows recipients to readily determine when information conveyed by the protocol should and should not replace information already received, and largely avoids the need to wait for timeouts to age out old information.

2.6. KSP Transmission

KSP transmissions are rate limited and, if a station has received proof of liveness from more than one other station, subject to a small random jitter. Normally transmissions occur at roughly ½ second intervals, but the rate limiter allows a short burst, sufficient for connectivity to be established without any timer delay for a point-to-point CA.

If delay bound support is not required the periodic transmission rate can be reduced to one every 5 seconds.

2.7. SAK agreement

This revision of this specification supports two different methods of choosing SAKs. In the first, "SAK calculation"(2.8), all the participants independently apply a pseudo-random function that uses the secret CAK to the key contributions publicly distributed by all live participants. In the second, "SAK distribution", the CA Leader—chosen by its SCI and MI—distributes the SAK to the other participants.

In both methods the key contribution KC is changed by a participant whenever it wants to prompt for the distribution of a new key, although a CA Leader will attempt to anticipate the need for new keys arising from simple exhaustion of MACsec PN space by any participant. In both cases each participant maintains a record of all the SAKs calculated while it is using a given KC[19] value, together with the highest Packet Number (PN) used as part of the MACsec IV when that SAK is protecting a MACsec frame. A new KC is randomly chosen when the participant is reinitialized and when the PN space for any of the derived SAKs is close to exhaustion. A new KC is also chosen whenever the CAK or Member Identifier is changed.

2.8. SAK calculation

When the SAK calculation method is used, each KSP participant independently calculates SAKs as a pseudo-random function of the CAK and the 128-bit KCs from all active peers.

The proposed prf is a hash that uses the CAK and the sequence of octets obtained by concatenating all the KCs together in the numeric order of the Member Identifiers of the contributing participants, with the greatest first. This revision suggests that the prf be the essntially the same as that used to generate the ICV for integrity protecting KSPUs. Suggestions for a better prf are most welcome.

Specifically:

SAK = AES-CMAC(K,M,128)

Where M comprises the concatenated KCs, and K is derived from the CAK as follows:

K = AES-ECB(CAK,0x2)

where the CAK is the AES key and the encrypted data is a single 128-bit block with the value '0x2'[20].

[19] It is an illusion that use of central distribution removes this requirement. If a participant simply forgot prior keys when a Leader issued a new key it would be easy to mount an attack in which the participant was first allowed to see frames from one Leader, then only from another, and then from the first again. In the absence of memory of prior keys and of a protected acknowledged method for ensuring that a new key has been distributed, such a participant could be induced to repeat the use of an SAK, PN tuple. Even if the distributed SAK is not a function of the KCs, requiring that a Leader change the SAK when a new KC is seen and reflecting that KC back to the participant in the KSPDU with the distributed SAK ensures that the participant knows what history has to be retained, and gets a chance to purge that history.

[20] An explanation of why it is necessary to ECB the CAK before using it in CMAC would be useful. I think it is because the CAK has also been used in the AES Key Wrap, and the ECB is being used to derived subkeys from the CAK rather than relyiing on any

A separate Key Identifier (KI) for the key is calculated as the simple exclusive-or of the KCs from active peers. Two KSP participants know, to within a very high probability, that they have agreed on the SAK if they are both advertising the same Key Identifier, but the KI provides no information to an attacker that could not be otherwise learnt from observing KSPDUs. In the absence of knowledge of the CAK, an attacker has no way to calculate the SAK using the KCs or KI. Moreover the attacker does not know the relationship between the SAKs produced by selectively removing KCs from the calculation by blocking some of the KSP communication.

2.9. SAK distribution

When the SAK distribution method is used, any participant that has one or more live peers and considers itself to be the CA Leader, or has no live peers but has potential peers and considers itself to be the CA Leader even if those potential peers were live, will distribute a randomly chosen SAK[21].

The SAK and an accompanying Key Identifier KI are distributed in the optional key distribution TLV in KSPDUs. These have a type code 0x01 (in 2 octets), followed by a length of 0x04 (in two octets), followed by the KI (in 8 octets), followed by the AES Key Wrap protected SAK in 24 octets. The only wrapped data is the SAK itself.

The key wrap Key Encrypting Key (KEK) is a sub-key derived from the CAK as follows:

KEK = AES-ECB(CAK, 0x0)

where CAK is an AES key, and the encrypted data is a single 128-bit block with the value ‘0x0’. The AES Key Wrap default IV defined in [10] MUST be used[22].

If the CA Leader itself decides to distribute a new key it should change its own KC, and just as for the SAK calculation method, the KI distributed should be the exclusive-or of all the KCs. There is no dependency on this choice of KI, and it could be randomly chosen, but using the suggested value makes it easier to debug protocol operation and has no security downside.[23]

2.10. SAK installation and use

When using KSP each MACsec participant transmits and receives using at most two SAKs at a time, no matter how many members a CA has. Two SAKs are required to ensure continuous connectivity when one changes. Different participants may change to a new key for transmission at different times and can have already protected frames buffered locally or within the bridges that support a virtual LAN.

When the participants in the CA and their KCs have not changed for a while, they will all be transmitting and receiving using the same SAK and will have the spare resource to devote to a new SAK should one be required, as will a participant that has just be initialized or attached to the LAN.

Given the available resources, any in use SAK is advertised in the actor’s KSPDU as the Old Key Identifier (OKI) together with the MACsec Association Number (AN) used together with the MACsec SCI to identify the SA used to transmit data frames protected with the SAK, and the Latest Key Identifier (LKI) is used to advertise the KI calculated using the current KCs from the actor and all active peers (or the KI distributed by the CA Leader if key distribution is being used).

A new SAK corresponding to the current LKI value will be calculated and installed, i.e. submitted to the MACsec entity for any precalculation of tables and configuration of hardware that may be required, when the SAK calculated from the KCs of all active peers differs from that currently in use, and either:

a) there is at least one active peer, no potential peers, and there is no previous key being used for receive or transmit; or

b) all active peers are advertising the same LKI as calculated by the actor.

At this point the actor assigns the next available AN to the new SAK. Once the key has been installed, the potential MACsec SAs are installed — specifying the receive SCI, AN, and acceptable Packet Numbers (PNs) for each of the active peers — and a KSPDU is sent indicating that the actor is prepared to receive using the key corresponding to the LKI.

When all active peers have indicated their ability to receive using the new SAK it can be used for transmission. Once a new key is used to transmit, KSPDUs are used to indicate that reception is disabled for the prior key (if any) and the key is uninstalled after a short delay sufficient for reception of any frames already transmitted, and for the other members of the CA to start using the new key. The station machine of Figure 4 specifies the life cycle of each SAK.

2.11. Data delay protection

Along with each of its key identifiers, the LKI and OKI, each participant advertises lowest

proof that two different direct uses of the CAK can never cause an exposure.

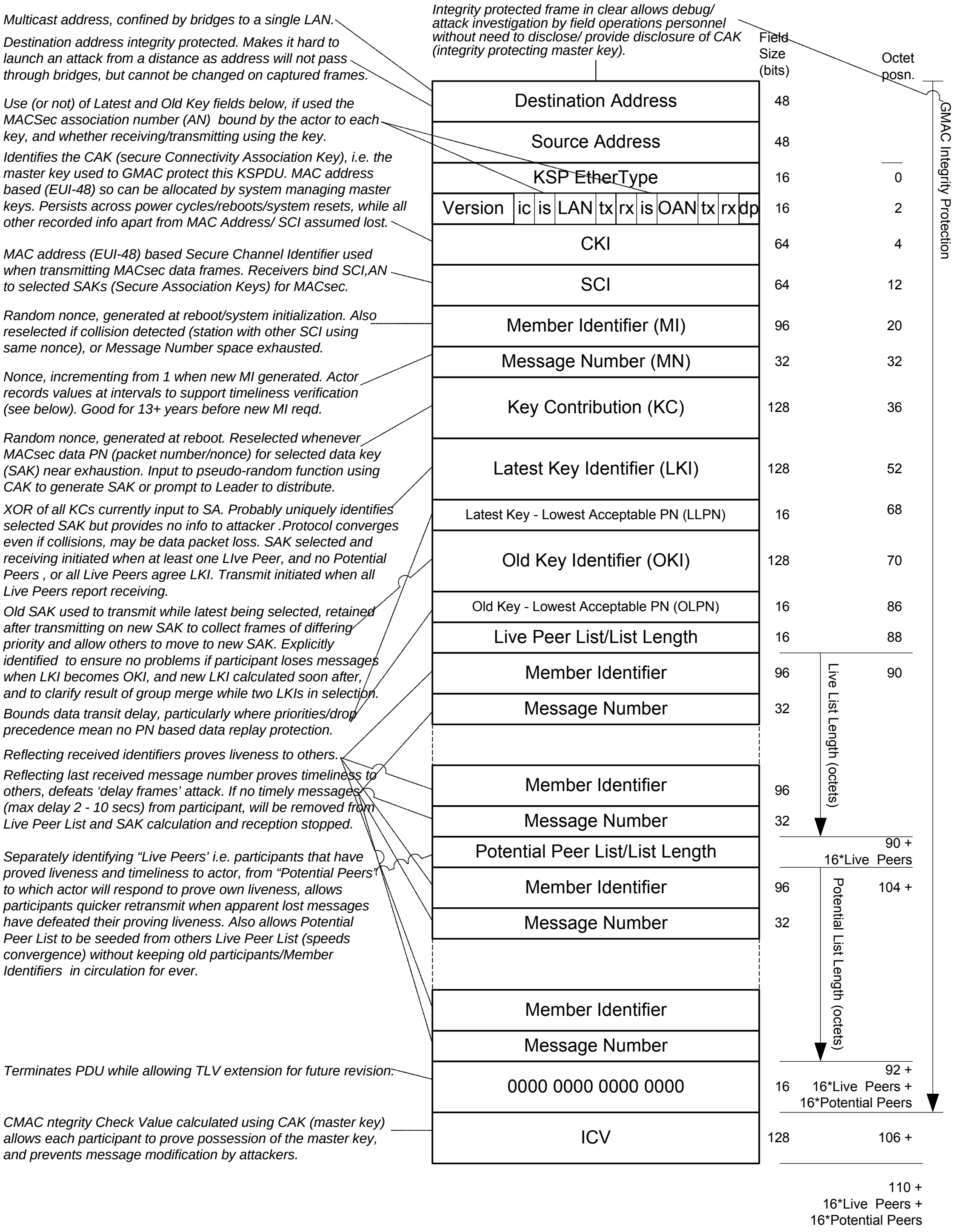
[21] Better words than “randomly chosen” are required here. The usual stuff. Unguessable, even with the knowledge of all past history.

[22] This part of the specification borrowed from Brian Weis’ LKS specification.

[23] The SAK distribution method does not provide as tight a guarantee that a fresh key has indeed been delivered as does the SAK calculation method. Each participant needs to check that the Leader considers itself to be live, and needs to retain the knowledge of a past key for at least the maximum period for which it considers the Leader to be live after the participant has changed the KC it was currently using when that key was distributed. With the SAK calculation method keys based on old KCs could be discarded as soon as that KC was discarded.

acceptable packet numbers (LLPN, OLPN) for each of the keys. Each of these reflects the lowest PN used within a one second window. Each of the participant's peers uses the time bounds provided by its proof of liveness, through reflection of their MI.MN tuples, together with the LPN values to discard delayed traffic.

Enforcement of delay bounds necessitates transmission of KSPDUs at frequent (½ second) intervals, to meet a maximum data delay of two seconds while minimizing the chance of connectivity interruption due to the possibility of lost or excessively delayed KSPDUs. KSP can operate without data delay protection, lessening the receive processing requirements in large CAs. However one of the ways to disrupt overall network stability is to attack the configuration protocols that MACsec is designed to protect by alternately delaying and delivering their PDUs, typically with cycle times in the range 4-30 seconds. Such attacks can cause effects that go beyond the immediate LAN. If data delay protection is not used, other procedures should be used to minimize the opportunity for such an attack.



Integrity protected frame in clear allows debug/ attack investigation by field operations personnel without need to disclose/ provide disclosure of CAK (integrity protecting master key).

Multicast address, confined by bridges to a single LAN.

Destination address integrity protected. Makes it hard to launch an attack from a distance as address will not pass through bridges, but cannot be changed on captured frames.

Use (or not) of Latest and Old Key fields below, if used the MACSec association number (AN) bound by the actor to each key, and whether receiving/transmitting using the key.

Identifies the CAK (secure Connectivity Association Key), i.e. the master key used to GMAC protect this KSPDU. MAC address based (EUI-48) so can be allocated by system managing master keys. Persists across power cycles/reboots/system resets, while all other recorded info apart from MAC Address/ SCI assumed lost.

MAC address (EUI-48) based Secure Channel Identifier used when transmitting MACsec data frames. Receivers bind SCI,AN to selected SAKs (Secure Association Keys) for MACsec.

Random nonce, generated at reboot/system initialization. Also reselected if collision detected (station with other SCI using same nonce), or Message Number space exhausted.

Nonce, incrementing from 1 when new MI generated. Actor records values at intervals to support timeliness verification (see below). Good for 13+ years before new MI reqd.

Random nonce, generated at reboot. Reselected whenever MACsec data PN (packet number/nonce) for selected data key (SAK) near exhaustion. Input to pseudo-random function using CAK to generate SAK or prompt to Leader to distribute.

XOR of all KCs currently input to SA. Probably uniquely identifies selected SAK but provides no info to attacker .Protocol converges even if collisions, may be data packet loss. SAK selected and receiving initiated when at least one LIve Peer, and no Potential Peers , or all Live Peers agree LKI. Transmit initiated when all Live Peers report receiving.

Old SAK used to transmit while latest being selected, retained after transmitting on new SAK to collect frames of differing priority and allow others to move to new SAK. Explicitly identified to ensure no problems if participant loses messages when LKI becomes OKI, and new LKI calculated soon after, and to clarify result of group merge while two LKIs in selection.

Bounds data transit delay, particularly where priorities/drop precedence mean no PN based data replay protection.

Reflecting received identifiers proves liveness to others.

Reflecting last received message number proves timeliness to others, defeats ‘delay frames’ attack. If no timely messages (max delay 2 - 10 secs) from participant, will be removed from Live Peer List and SAK calculation and reception stopped.

Separately identifying “Live Peers’ i.e. participants that have proved liveness and timeliness to actor, from “Potential Peers” to which actor will respond to prove own liveness, allows participants quicker retransmit when apparent lost messages have defeated their proving liveness. Also allows Potential Peer List to be seeded from others Live Peer List (speeds convergence) without keeping old participants/Member Identifiers in circulation for ever.

Terminates PDU while allowing TLV extension for future revision.

CMAC ntegrity Check Value calculated using CAK (master key) allows each participant to prove possession of the master key, and prevents message modification by attackers.

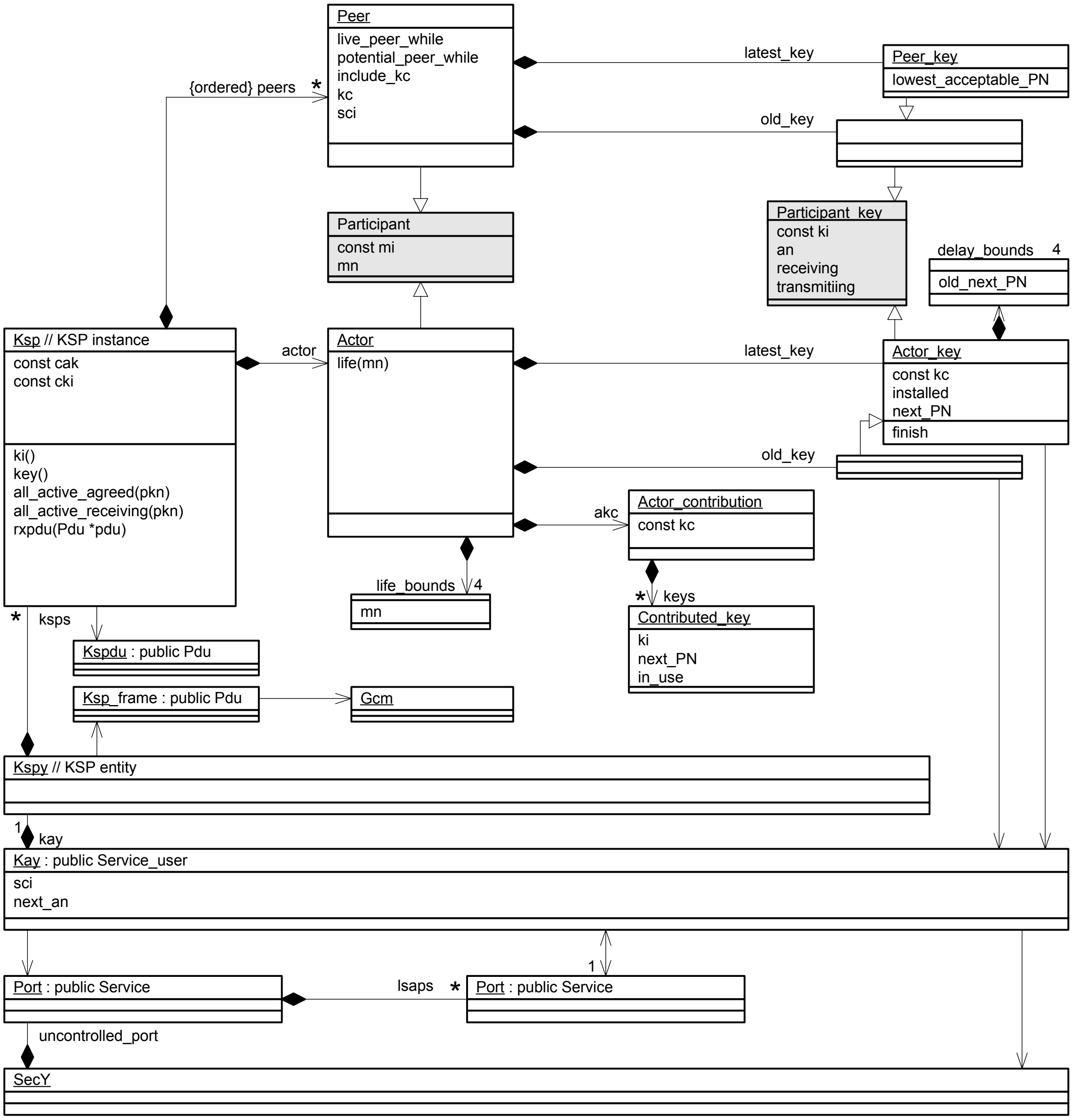
Figure 1 — KSPDU Format and Fields

3. Model of operation

Figure 2 is an object model of part of a KSP participant, including the KSP Entity itself and its relationship to the MACsec Entity (SecY). Figure 3 (tbs) summarizes the notation used, which follows UML 2.0 conventions.

The KSP Entity forms part of the Key Agreement Entity, which in turn attaches to the Uncontrolled Port provided by the SecY. Each KSP Entity can support a number of KSP instances, each with a fixed CAK and corresponding CKI. Each KSP instance maintains its own actor and list of peers.

Throughout the diagram objects with 'const' attributes retain those attributes throughout their life. Changing those attributes requires destruction and recreation of those objects. Thus changing the actor's key contribution (kc) discards the records of keys to which the kc has contributed. Similarly changing the actor's member identifier results in resetting the message number and choosing a new key contribution. The records of keys generated with the key contribution are not required unless further keys are to be generated with the past contribution, so either of the keys in use can continue to be used until a fresh key with the new key can be brought into service.



KSP Objects <<KSPO 0.3>>

Figure 2 — KSP Object Model

Figure 3 — object diagram notation

This section specifies the operation of the KSP Entity with reference to the object model of Figure 2. The principal functional aspects of operation are first described in relation to state machines implemented by the objects, this description is then followed by a detailed specification of the variables and procedures used by each object. Conformance to the specification requires implementation of externally observable behavior that corresponds to the behavior of the specified objects and their variables and procedures, including procedures whose behavior is specified as corresponding to that of the specified state machines.

3.1. State machines

The following objects implement the following state machines, in addition to procedures that perform simple processing or supply information to other objects:

- Ksp — Ksp instance
    - Receive Machine
    - Transmit Machine
    - Actor Creation Machine — instantiates and refreshes actor (Actor object) instances
- Actor
    - Key Generation Machine — instantiates new keys (Actor_key object) and instantiates and refreshes the actor's key contribution (Actor_contribution object)
- Actor_key
    - Actor Key Machine

3.2. Receive KSPDU processing

The MAC Security Entity (SecY) notionally provides all received frames to the users of its Controlled Port. The Key Agreement Entity (KaY) uses the services of an LLC Entity[24], selecting only those it is interested in as identified by their Ethertypes, including KSP[25] frames. These are submitted to the KSP Entity (Kspy).

The KSP Entity first checks that the destination MAC address is the assigned multicast address. Then it extracts the CKI from the frame to identify the KSP instance (Ksp object) that is the intended recipient and its CAK. GMAC (GCM Integrity Check) is used to validate the frame. If successful the received frame is marked as valid, the addresses and ICV removed, and it is passed to the Ksp's Receive Machine (Figure 4). This counts and discard invalid frames, and processes the remainder as described below.

If the frame's[26] MI and SCI are the same as those of the (receiving) actor, then the frame has, in all probability, been loop backed to the transmitter[27] and is counted and discarded. If just the MI is the same it seems that the unlikely event (1 in $2^{90 \text{ odd}}$) of a duplicate MI has happened. A count is incremented, the PDU discarded, and the actor's own MI changed by deleting and recreating the Actor object and its dependents This will disrupt connectivity if the KSP instance is controlling the MACsec keys. The most likely cause of duplicate MIs is a broken pseudo-random number generator, so the counted event warrants investigation.

The actor's list of peers is searched for a match with the frame's MI. If the peer is found then its recorded MN is compared with the frame's MN, and misordered or duplicated frames are counted and discarded. The frame's SCI is checked against the recorded SCI recorded. If it has changed the peer record is deleted, and processing proceeds as if the frame were the first received from the peer. A peer not found in the list is added to it. Since key contributions from live peers on the list are used ordered by MI to calculate an SAK it is convenient, though not essential, to keep the list ordered.

The potential life of the peer, i.e. the time the peer record will be kept, (potential_peer_while) is updated even though the peer has not yet been checked for liveness. If frames from a past peer are being replayed by an attacker the minimum work should be done for each frame, and maintaining the peer in the list achieves that.

The peer record is updated with the receive MN, so that it can be used to check for misordered or duplicate frames with the same MI.

The frame's live and potential peer list are then scanned for the actor's own MI. If it is found the corresponding MN is compared to the actor's own records (life_bounds) of the time elapsed since that MN was transmitted, and the time for which the peer record should be considered live (live_peer_while) is calculated.

If the peer is live then its key contribution may be newly included or changed, forcing a key recalculation. A live peer may not wish to contribute to a new key, as indicated by the include_kc flag in the KSPDU : it may have been sent to play out the delay protection for frames currently sent by a participant intent on calculating new keys with a different KSP instance and a more recent SAK.

Each participant indicates whether receivers should apply delay protection to received data. If delay protection is off, the peer is not guaranteed to transmit KSPDUs sufficiently frequently to allow delay bounds to be imposed.

[24] In keeping with recent 802.1 practice an LLC Entity is considered to include Ethertype multiplexing/demultiplexing.

[25] In keeping with standards practice an Ethertype will not be assigned until the sponsor ballot stage is reached.

[26] Throughout this description the short hand "the frame's MI" is means "the MI used by the participant transmitting the frame".

[27] Active loopback is a curse and can easily stop a network working. The damage done far exceeds its diagnostic potential.

The receive key information for a live peer is updated, including the keys used (as indicated by the LKI and OKI key identifiers), their associated SAs as identified by their association numbers (AN), and lowest acceptable packet numbers used to enforce delay protection.

The peers included in the live peer list included in a KSPDU sent by a live peer are added to the actor's own list of peers if not already present.

If the current KI is to be recalculated as a result of receiving the KSPDU then the actor's Actor Key Machines for both the latest and old key are executed, as is the Key Generation Machine. C++ code for receive processing follows (Figure 5).

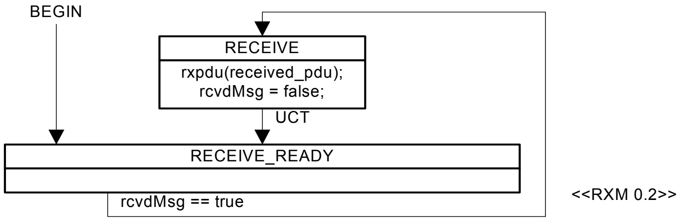


Figure 4 — Receive machine

```
void Ksp::rxpdu(Pdu *received_pdu)
{
   Kspdu  rcvd(received_pdu);

   if (!rcvd.valid )      { rcv_event(Invalid_pdu)    return; };
   if ((rcvd.sci == sci) && (rcvd.mi  == actor.mi))
                          { rcv_event(Loopback_pdu)   return; };
   if ( rcvd.mi == actor.mi)
   {
      this.change_mi();     rcv_event(Duplicate_mi)   return;
   };                                // broken psrng?

   Peer *peer = find_peer( rcvd->mi);
   if (peer != 0)
   {
      if (rcvd.mn  <  peer->mn) { rcv_event(Misordered_pdu) return; };
      if (rcvd.mn  == peer->mn) { rcv_event(Duplicate_pdu)  return; };

      if (rcvd.sci != peer->sci){ rcv_event(Peer_sci_changed);
                                 delete peer; peer = 0;           };
   };
   if (peer == 0)
   {
      peers.push_back(Peer( this, rcvd.mi, rcvd.sci));
      peer = &(peers->last());
   };

   peer->potential_peer_while = potential_peer_life;
   peer->mn  =  rcvd->mn;

   Ticks life = actor->life(rcvd->find_me(actor->mi));
   if (life > peer->live_peer_while) peer->live_peer_while = life;

   if (peer->live_peer_while != 0)
   {
      if ((peer->include_kc != rcvd->include_kc) || (peer->kc != rcvd->kc))
         bool recalculate_key = true;

      peer->include_kc    = rcvd->include_kc;
      peer->kc            = rcvd->kc;

      peer->delay_protect = rcvd->delay_protect;

      peer->rx_keys(&rcvd);

      add_potential_peers(*(rcvd->peers)); // from live peer's live list

      if (recalculate_key)
      {
         if (old_key != 0)    old_key->actor_key_sm();
         if (latest_key != 0) latest_key-> actor_key_sm ();
         key_generation_sm();
      };
}; };
```

Figure 5 — KSPDU receive processing

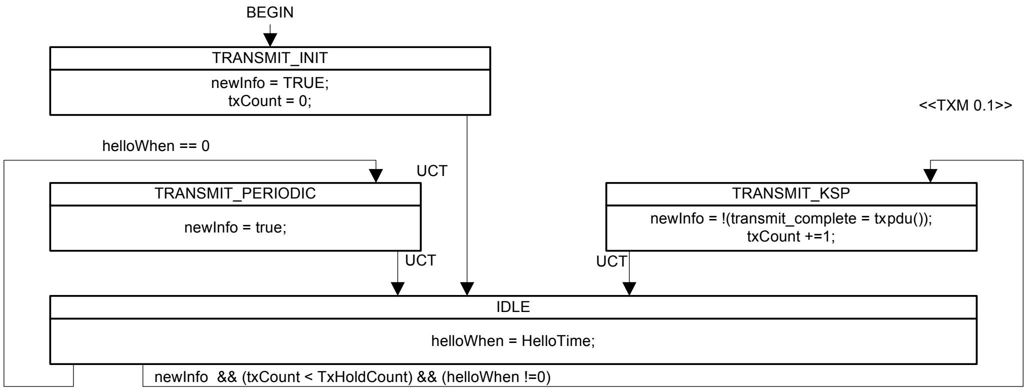
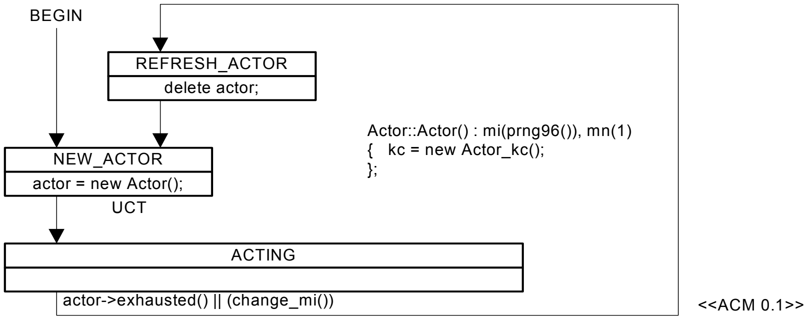


Figure 6 — Transmit state machine



```
Actor::Actor() : mi(prng96()), mn(1)
{   kc = new Actor_kc();
};
```

Figure 7 — Actor Creation state machine

3.3. Transmitting KSPDUs

Each KSP instance transmits KSPDUs independently of each of the others, subject to its own transmission rate limiter. The transmit machine, implemented by the Ksp object, is specified in Figure 6. Each of the other machines, if it detects a need to transmit new information to other KSP participants, sets the newInfo flag to prompt a transmission. In addition the transmit machine ensures that periodic transmissions take place.

If the number of participants is limited (to 86 or less), information for the actor, live peers, and potential peers is included in a single KSPDU. Otherwise the actor's own information is sent in every PDU, but the live and potential peer lists are filled by proceeding through the Ksp's list of peers, recommencing with the next KSPDU.

3.4. Choosing and changing MIs

Each KSP Instance uses a single Member Identifier (MI) at a time. However the value may need to be changed, first to handle the unlikely event of a collision of choices by different participants. Secondly the MI has a limited life, since the MI.MN tuple must never be repeated[28]. At the envisaged maximum transmission rate of 10 KSPDUs per second it is conceivable that the MI would have to be changed once every ten years. The 96 bit MI is chosen at random, i.e. such that the distribution of MIs chosen by any set of KSP participants using the same implementation of MI choice is indistinguishable from a random selection.

The MI[29] effectively defines a multipoint connection from the actor to each of its peers, ordering the KSPDUs transmitted. This allows each change of Key Contribution (KC) by the actor to be communicated effectively. When the MI is changed there will be a short period when the actor's peers will have a record of its previous MI and KC, and until this times out a new key value will not be agreed. Note that the actor's KC is within the scope of the Actor object, which is recreated when the MI changes so that the KC will be forced to change at the same time. This avoids the possibility of a single participant contributing the same KC twice, and thus having it cancel out in the KI calculation.

Figure 7 specifies the actor creation state machine implemented by the Ksp object.

3.5. Key Generation

The actor object implements the key generation machine (Figure 8). A new key (SAK) is generated if the actor has active peers, there is currently only one key (i.e. no old key), the key generation machine has not been told to finish (in deference to another KSP instance),the key selected is not already in use and its PN space is not exhausted. The existing Actor_contribution is deleted and recreated if a new key is wanted but its PN space is exhausted.

The Key Generation machine does not delete the keys it creates. The instance of the actor key machine created with each key does this after the key is no longer used for transmission or reception.

3.6. Key installation and use

Each Actor_key is created by the Key Generation machine as specified by the C++ code in Figure 9, and implements the actor key machine of Figure 10 and previously described in section 2.8.

[28] At least the chance of repeat with the same CAK must be so rare that no attacker will ever think of looking for one.

[29] It might be thought that the MAC address based SCI could serve as a sufficient unique identifier of a system, in combination with the KC itself. However duplication of MAC addresses, both accidental and deliberate, is far more common than it should be and attempting to guess when this is occurring is much more difficult if the protocol lacks the explicit MI. Trying to economize on fields in PDUs is a bad choice. KSP spots duplicate SCIs reliably.

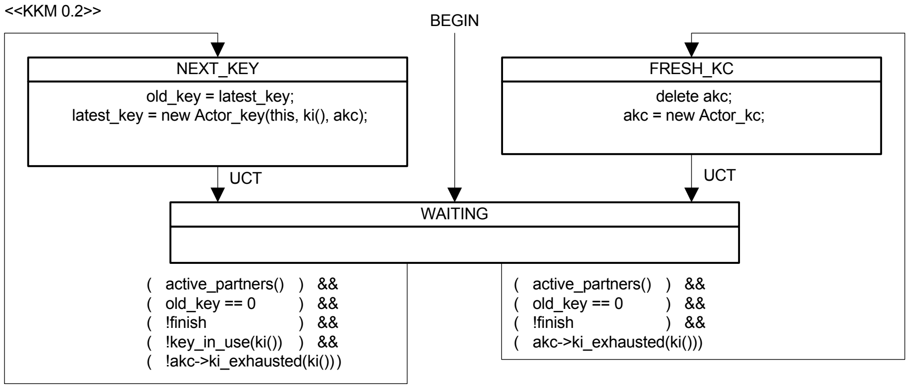


Figure 8 — Key Generation state machine

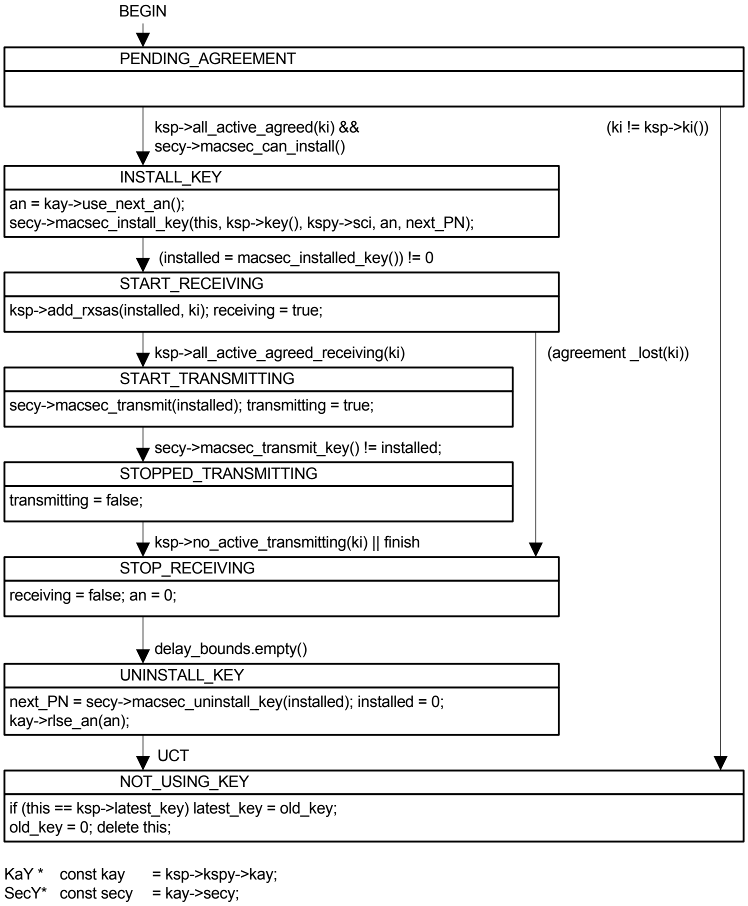
```
Actor_key::Actor_key(Ksp *p, KI key_id, KC key_contribution) : Participant(key_id), ksp(p),
kc(key_contribution),

    receiving = transmitting = finish = false; installed = 0; an = 0;
   next_PN   = ksp->next_pn_for(key_contribution, key_id);

   for (int i = 0; i < ticks_to_record; i++) delay_bounds.push(next_PN);

   akm = PENDING_AGREEMENT;
   dbm = DELAY_BOUND;
};
```

Figure 9 — Actor key creation


BEGIN
PENDING_AGREEMENT
ksp->all_active_agreed(ki) &&
secy->macsec_can_install()
(ki != ksp->ki())
INSTALL_KEY
an = kay->use_next_an();
secy->macsec_install_key(this, ksp->key(), kspy->sci, an, next_PN);
(installed = macsec_installed_key()) != 0
START_RECEIVING
ksp->add_rxsas(installed, ki); receiving = true;
ksp->all_active_agreed_receiving(ki)
(agreement _lost(ki))
START_TRANSMITTING
secy->macsec_transmit(installed); transmitting = true;
secy->macsec_transmit_key() != installed;
STOPPED_TRANSMITTING
transmitting = false;
ksp->no_active_transmitting(ki) || finish
STOP_RECEIVING
receiving = false; an = 0;
delay_bounds.empty()
UNINSTALL_KEY
next_PN = secy->macsec_uninstall_key(installed); installed = 0;
kay->rlse_an(an);
UCT
NOT_USING_KEY
if (this == ksp->latest_key) latest_key = old_key;
old_key = 0; delete this;
KaY * const kay = ksp->kspy->kay;
SecY* const secy = kay->secy;

Actor key state machine <<AKM 0.3>>

Figure 10 — Actor key machine

3.7. Kspy

This subsection, and others like it, will be eventually provided to detail the variables and procedures supported by each of the objects that compose the operational model.

4. Examples

This section provides some examples of KSP operation, focusing on the essentials of key agreement, installation, and use. Each message is shown with its three component parts — actor information, live peer list, and potential peer list — separated by the | symbol thus:

Actor | Live list | Potential list

and each MI + MN tuple as X+1, X+2, etc. where X is the MI value.

4.1. Two participants

Consider two stations S_A, S_B each with an MI+MN of A+.., B+.., and key contributions of KC_A, KC_B. The KSPDU exchange following initialization of the stations proceeds as follows:

S_A→A+1, KC_A || →S_B.. (1)

S_B→B+1, KC_B || A+1 →S_A.. (2)

S_A can now calculate SAK_{AB} with key identifier KI_{AB}, since S_B has proved itself live and S_A knows of no other potential participants. Assuming that S_A waits to turn its receiver on to receive from S_B before transmitting again:

S_A→A+2, $KC_{A,}$, LKI_{AB}.r | B+1 | →S_B.. (3)

S_B can now receive and transmit using the agreed key, and transmits:

S_B→B+2, $KC_{B,}$ LKI_{AB}.rt | A+2 | →S_A.. (4)

so A can start receiving and transmitting.

Thus in the point-to-point case, in the absence of an attack, KSP's behavior is the same as that of the well-known 4-way handshake.

4.2. Another participant joins

Continuing with the prior example, a station S_C is attached to the LAN or powered up. Assuming, in order to be explicit about the message exchanges, that S_C is just in time to receive the last message of the previous sequence:

S_B→B+2, $KC_{B,}$ LKI_{AB}.rt | A+2 | →S_A, S_C.. (4)

then S_C will record both S_A and S_B as potential peers and transmit:

S_C→C+1, KC_C || A+2, B+2 →S_A, S_B.. (5)

this message will prove S_C's liveness to both S_A and S_B who will independently move SAK_{AB} to being their old key, calculate a new key SAK_{ABC} with KI_{ABC}, and transmit:

S_A→A+3,KC_A,LKI_{ABC},OKI_{AB}.rt|B+2,C+1| →S_B,S_C(6)

S_B→B+3,KC_B,LKI_{ABC},OKI_{AB}.rt|A+2,C+1| →S_A,S_C(7)

Neither of these two messages will cause S_A or S_B to transmit again (until their periodic transmit timers elapse of course) as they have no new status to report, nor will S_C transmit until it has received both. Once S_C receives the messages it will have proof of S_A and S_B's liveness and will have calculated the same LKI as they have, so it will install SAK_{ABC}, enable reception, and transmit:

S_C→C+2, KC_C,LKI_{ABC}.r | A+3,B+3 | →S_A,S_B(8)

S_A and S_B will then install the key and enable reception, and transmit:

S_A→A+4, $KC_{A,}$,LKI_{ABC}.r, OKI_{AB}.rt | B+3,C+2 |→S_B,S_C(6)

S_B→B+4, KC_B,LKI_{ABC}.r, OKI_{AB}.rt | A+4,C+2 |→S_A,S_C(7)

whereupon all the participants can transmit. A little later their periodic transmissions will show that they are receiving and transmitting using LKI_{ABC}, but these at not required to establish the new connectivity.

In general addition to a group requires each participant to send two messages.[30] More messages can be transmitted if some or all of the participants have a significant transmission delay after processing.

4.3. Forcing a key change

The members of a CA transmit data independently at their own rate, so it is not known in advance which will come close to exhausting the PN space for an SAK first. Any KSP participant can force a key change by changing its KC. Assuming two participants, that having been contributing KC_A, KC_B for a while, the last periodic messages transmitted prior to the change will be:

S_A→A+3, $KC_{A,}$ LKI_{AB}.rt | B+5 | →S_B.. (1)

S_B→B+6, $KC_{B,}$ LKI_{AB}.rt | A+3 | →S_A.. (2)

(where the numbers '3' and '6' and their successors are used as short hand for what will be much larger numbers — the time between forced key changes is at least 5 minutes for 10 Gb/s Ethernet and likely to be over a month for 100 Mb/s in typical use).

Assuming S_A needs to change the key first, it generates a new key contribution KC_{A2}, calculates LKI_{A2B}, and transmits:

S_A→A+4, $KC_{A2,}$ LKI_{A2B}, OKI_{AB}.rt | B+6 | →S_B.. (3)

On receipt, S_B calculates the same new SAK and LKI, installs the new key, enables reception, and transmits:

S_B→B+7, $KC_{B,}$ LKI_{A2B}.r, OKI_{AB}.rt | A+4 | →S_A.. (4)

On receipt of this message S_A enables reception, starts transmitting data, and transmits:

S_A→A+5, $KC_{A2,}$ LKI_{A2B}.rt, OKI_{AB}.rt | B+7 | →S_B.. (3)

so S_B can start transmitting data using the new key, completing the change.

In general a key change requires each participant to transmit a single message, plus one message for the initiator of the change

[30] My October presentation described key installation and reception enabling criteria that require only a single message from each participant. These have a slightly higher chance of flapping connectivity for the new participants if many join at the same time i.e. within a 100 milliseconds or so. Simulation may show which strategy is to choose. In principle participants can make their choice independently, as a combination is interoperable, but the specification should make the definite choice.

4.4. Message crossing

There is nothing particular about the choice of S_A and S_B in the two participant startup example above (4.1), so the message exchange described is clearly independent of which station transmits first. It is possible that they transmit at the same time, or at least before either has processed its receive message. It is a good idea in KSP, and a number of protocols, to process any received messages when scheduled before transmitting, but the 'message crossing' can still occur. KSP will still complete promptly, with the worst case of message crossing proceeding as follows:

S_A→A+1, KC_A || →S_B.. (1)

S_B→B+1, KC_B || →S_A.. (2)

S_A→A+2, KC_A || B+1 →S_B.. (3)

S_B→B+2, KC_B || A+1 →S_A.. (4)

S_A→A+3, $KC_{A,}$, LKI_{AB}.r | B+2 | →S_B.. (5)

S_B→B+3, KC_B, LKI_{AB}.r | A+2 | →S_A.. (6)

At this point both S_A and S_B can receive and transmit using SAK_{AB}, so instead of the required 4 messages a total of 6 have been sent. The transmit state machine introduces a small random (in the weak sense) delay whenever a station has more than one active peer. This reduces the expected number of messages sent for large group CAs.

4.5. Multiple key changes

There is a small chance that two participants will decide to revise their key contributions at the same time, though a change by either would result in a new key that would have solved the other's PN space problem. With two participants that have just sent the messages:

S_A→A+3, $KC_{A,}$ LKI_{AB}.rt | B+5 | →S_B.. (1)

S_B→B+6, $KC_{B,}$ LKI_{AB}.rt | A+3 | →S_A.. (2)

and then both decide to change their key contributions, the sequence of messages proceeds as follows.

S_A→A+4, $KC_{A2,}$ LKI_{A2B}, OKI_{AB}.rt | B+6 | →S_B.. (3)

S_B→B+7, $KC_{B2,}$ LKI_{AB2}, OKI_{AB}.rt | A+3 | →S_A.. (4)

(3) →S_B.......

at which point both S_A and S_B will calculate a new key with LKI_{A2B2} and transmit once more, assuming S_A happens to transmit first:

S_A→A+5, $KC_{A2,}$ LKI_{A2B2}, OKI_{AB}.rt | B+7 | →S_B.. (5)

now, from S_B's perspective, all active participants have agreed on the same key so it can be installed and reception enabled:

S_B→B+8, $KC_{B2,}$ LKI_{A2B2}.r, OKI_{AB}.rt | A+5 | →S_A.. (6)

and on receipt S_A can do the same, and switch transmission to the new key, transmitting:

S_A→A+6, $KC_{A2,}$ LKI_{A2B2}.rt, OKI_{AB}.r | B+8 | →S_B.. (7)

allowing S_B to switch transmission to the new key. Eventually reception using the old key will stop, and it will be uninstalled.

4.6. Participant leaves

Say that S_A, S_B, and S_C have agreed SAK_{ABC} based on their key contributions KC_A, KC_B, KC_C, and have just transmitted messages:

S_A→A+3, KC_A,LKI_{ABC}.rt|B+3,C+3| →S_B, S_C(1)

S_B→B+3, KC_B,LKI_{ABC}.rt|A+3,C+3| →S_B, S_C(2)

S_C→C+3, KC_C,LKI_{ABC}.rt|A+3,B+3| →S_B, S_C(3)

(where '3' stands for whatever MN S_A, S_B, S_C, have individually reached thus far for their MIs) when S_C is removed from the LAN. S_A and S_B will carry on with the same key for a brief while, until one or the other of them times out S_C. Assuming S_A does so first, after two further periodic transmissions. It will transmit:

S_A→A+6, KC_A, LKI_{AB},OKI_{ABC}.rt|B+5| →S_B.. (4)

Though S_B might send

S_B→B+6, KC_B, LKI_{ABC}.rt|A+6,C+3| →S_A.. (5)

at its next transmission, but it will also eventually time out S_C, and will then send:

S_B→B+7, KC_B, LKI_{AB}.r,OKI_{ABC}.rt|B+5| →S_A.. (6)

since S_A has already agreed LKI_{AB}. On receipt S_A can install SAK_{AB}, enable reception, start transmitting data using the key, and transmit:

S_A→A+7, KC_A, LKI_{AB}.rt,OKI_{ABC}.r|B+7| →S_B.. (7)

which allows S_B to start transmitting using the new key.

The worst case of a new participant attempting to join a CA is when a previous participant has just left, since the existing participants will attempt to include the departed station for a while, but the newcomer cannot.

4.7. Agreement under replay attack

Consider the two participant startup example above (4.1). Assume that an attacker has eavesdropped on an earlier use of the LAN between S_A and S_B, and acquired one or messages from S_A when it was using the Member Identifier R with the present CAK — so the message will pass the GMAC integrity check. In some scenarios it may be very easy for the attacker to insert his equipment in the link between the two stations, fiber connections through a common patch panel in a collocation facility being one possible example. The attacker might wait until S_A transmits its first message:

S_A→A+1, KC_A || →S_B.. (1)

and then inject the replayed message, delivering it just to S_B.

→R+3, KC_R, LKI_{QR}.rt|Q+7| →S_B.. (1a)

Now S_B receives and processes both messages, recording both A and R (but not Q) as potential peers and transmits:

S_B→B+1, KC_B || A+1, R+3 →S_A.. (2)

this proves to S_A that S_B is live, or to be more exact it proves to the participant identified by A that a participant identified by B is live. S_A does not record R as a potential peer, since it is not in S_B's live list and transmits:

S_A→A+2, KC_A, LKI_{AB}.r | B+1 | →S_B.. (3)

having calculated SAK_{AB} using the key contributions of all active peers (i.e. itself and S_B). On receipt S_B finds that its calculation of the LKI agrees with that of all active peers, and therefore installs the key, enabling both reception and transmission, and transmits:

S_B→B+2, KC_B, LKI_{AB}.rt | A+2 | R+3 →S_A.. (4)

enabling S_A to start transmission.

This particular replay attack failed to increase the number of messages or time required for the true participants to start communicating.

4.8. Another replay attack

The attacker of the previous example tries again, this time sending the replayed message (1a) to both S_A and S_B.

→R+3, KC_R, LKI_{QR}.rt|Q+7| →S_B.. (1a)

Assuming that S_A processes this message before S_B sends, the message sequence proceeds as follows.

S_A→A+2, KC_A || R+3 →S_B.. (2)

S_B→B+1, KC_B || A+1, R+3 →S_A.. (3)

S_A→A+3, KC_A, LKI_{AB} | B+1 | R+3 →S_B.. (4)

S_B→B+2, KC_B, LKI_{AB}.r | A+3 | R+3 →S_A.. (5)

S_A→A+4, KC_A, LKI_{AB}.rt | B+1 | R+3 →S_B.. (6)

After receipt of this last KSPDU S_B can start transmitting. The sequence of messages has been extended by one because of the attack. additional replayed messages do not slow convergence on the key any more.

5. Goals and requirements[31]

KSP and the assumptions made about its placement within the overall key hierarchy for a network, attempt the following[32]:

1. Maximize the chance that the required full, i.e. symmetric and transitive, connectivity is provided between stations on the LAN.
2. Provide secure connectivity within a few seconds of the underlying LAN service becoming available.
3. Function correctly in an environment where stations are powered on and off at any time, and where an attacker may control power.
4. Allow any subset of stations to be powered off and on again without disrupting the connectivity[33] between the remainder of the stations.
5. Support both point-to-point and multipoint connectivity without preselection of one or the other.
6. Ensure delay bounds for MACsec data traffic.
7. Operate without requiring pairwise communication between all stations.
8. Allow connectivity to be re-established after power-up without requiring network connectivity to an authentication server.
9. Protects against attacks that attempt to exhaust resources by requiring difficult cryptographic calculations.
10. Minimize the number of SAKs that each station needs to support at any one time, in practice limiting these to two.
11. Operate without the use of computationally expensive public key cryptography techniques.
12. Not disclose the data keys (SAKs)

5.1. Non-goals

1. Guard against or compensate for the use of weak keys[34].

6. Target environment

KSP is designed to support fixed infrastructure connectivity requirements for enterprises and users of the P802.1ad Provider Bridge draft standard. In particular both the MEF's E-LINE (point-to-point) and E-LAN (multipoint-to-multipoint) services are supported. KSP is designed to work well for cases of point to point and small group connectivity, and the devices connected are typically fixed and stable in deployment. Over 90% of the CAs deployed are expected to be point-to-point, with the average group CA comprising 5 members, with very few groups of more than 30 members.

[31] Goals are those things that one would like to do and can do, the other things are non-goals. Requirements (in standards development) are things the other guy can't do.

[32] Amongst numerous other goals.

[33] It is not even necessary to change SAKs if each station has a real time clock of even modest accuracy.

[34] Possession of even a single KSP message allows an attacker to attempt an offline brute force attack. The KSP message is integrity protected so the attacker knows with reasonable certainty when the key has been guessed. Possession of a second KSP message confirms the key to a high probability. For this reason KSP is not used with easily guessable password based CAKs.

MACsec Summary

This section provides a brief overview of MACsec.

6.1. What MACSec does

MACsec secures a LAN. In many cases this means a physical point-to-point link. In others a number of LAN equivalents may be realized by multiplexing over a physically shared media. In another a virtual LAN may be provided at many customer sites by a provider bridged network. In all these cases the service provided by MACsec is that used by bridges and end stations. Securing a LAN is different from securing individual connections amongst stations connected to that LAN[35]. All that MACsec guarantees is that integrity and confidentiality will be preserved amongst the set of stations authorized to connect to the LAN.

An important goal of MACsec is not to change the way that bridges and routers work, while enabling them to apply policies[36] to the data that they forward. This means that MACsec itself should not interfere with the normal connectivity provided by a LAN to authorized stations.

6.2. Connectivity Associations

The authorized stations that are attached to a common LAN compose a MACsec secure connectivity association (CA), and prove their mutual authentication by exchanging messages that are integrity protected with a common master key, the CA Key (CAK).

The CAK may configured in each station out-of-band, using a local command line interface for example. Alternatively the CAK may be the direct or indirect result of executing a key agreement, key exchange, or authentication and authorization protocol. If the CA is point-to-point (i.e. has only 2 members) , the CAK may be the pair-wise master key (PMK) generated by EAP with one of the members as the EAP peer and the other as the authenticator. If the CA comprises three or more systems with a full or partial mesh of PMKs, a CAK can be assigned and distributed to each of the members using a trivial spanning tree protocol.

Since MACsec secures full (i.e. symmetric and transitive) connectivity between the members of a CA using symmetric key cryptography[37], all the members of the CA possess all the secure association keys (SAKs) used to support the CA. So the requirement for transitive connectivity ends by implying transitive trust within the CA – if A trusts B and B trusts C, then A necessarily trusts C. This transitive trust is captured by the single CAK. An attempt to enforce partial connectivity to match a partial mesh of master keys would cause client protocols to behave oddly, if not incorrectly. Equally the direct use of a mesh of keys opens up the prospect of complex accidental failures.

6.3. Secure Channels

In order to provide unique cryptographic nonces and replay protection, the data traffic from each transmitter in an CA is identified as belonging to a separate secure channel (SC). Each SC is supported by a succession of secure associations (SAs). One SA is replaced by another with a different secure association key (SAK) when the packet number (PN) space for an SA is close to exhaustion, or when the CAK is changed. SAKs may be unique to an SC, or shared amongst some or all SCs in the CA. KSP shares SAKs so that each participant in a CA only has to be able to receive and transmit using the same number of cryptographic keys as required for a point-to-point CA.

6.4. Changing Keys

KSP allows both SAKs and CAKs to be changed without disrupting the connectivity between stations, although one reason for changing a CAK is to create a new CA that excludes members of a prior CA.

While addressing the requirements of multipoint CAs, KSP is simple enough to be used unchanged if the CA is only point-to-point. This maximizes interoperability and helps considerably in those cases where a CA is initially believed to be point-to-point but turns out to be multipoint. KSP has to work well in an environment where stations are being powered up and down and different systems take more or less time to become functional after power up.

[35] If this is what is wanted from MACsec, then separately secured LANs need to be provided and interconnected with a trusted bridge or router.

[36] The policies should be based on the authorization accorded to the stations connected to each LAN

[37] For the picky it also has to be noted that MACsec does not use more than one key to integrity protect a frame, and that each frame handed to MACsec by its user is only sent once.