Using .1X/EAP for infrastructure authentication

Mick Seaman

This note is an attempt to set down some thoughts about the use of EAP and .1X to support P802.1af Key Agreement for MACsec protection of infrastructure links. At present it does not argue strongly for a particular conclusion, but sets out facts and opinions based on investigation to date. In that respect it is little more than an email, organized for easier editing and development, and is work in progress for discussion in the September 2005 802. 1interim.

1. Opportunities and problems

The issues raised here have been the subject of some prior .1 discussion, and have been briefly described in prior notes and presentations.

A large part of the motivation for MACsec is the ability to incrementally secure the weakest links within a network, quite possibly an bridged network, without imposing protocol specific end-to-end solutions. In that respect many aspects of its use are similar to the way in which .1X is currently used to secure point-to-point access links to a network. Given the widespread availability of the EAP and Radius infrastructure support for .1X, it is tempting to make use of that same support for MACsec, whether that is arguably the best technical solution or not. Discussion to date of the Key Agreement protocol to support MACsec has assumed that that protocol would begin by using a pre-shared "master" key (referred to as the CAK in P802.1AE), and it is easy to see that .1X EAP could be used to provide that CAK and associated data, just as it provides a master key for .11i derivation of temporary session keys. This is particularly true for MACsec CAs that only have two members, i.e. real or virtual point-to-point links. To simplify the discussion this note restricts itself to that point-to-point case for the present†1.

When infrastructure links are being supported it cannot be assumed that a connection to a back-end Authentication Server is available from either of the two parties at the time when the link becomes active. While it is possible to bring a network up by "growing" it from a core that contains the Authentication Server, such a network could be unacceptably slow to become operational after an outage. For this reason it is desirable to consider the distribution of a CAK†2, whose possession can be used to prove mutual authentication and to generate session keys (MACsec SAKs) known only to those possessing the CAK, as asynchronous with respect to the current operational state of the LAN and CA connectivity. Clearly a new CAK cannot be supplied by EAP when the two members of the CA have no connectivity over the LAN, but equally it is desirable that a previously distributed CAK be used by key agreement to generate SAKs so as to provided secure connectivity for data over the LAN in advance of obtaining a fresh CAK.

The assumption that EAP is to run every time LAN connectivity becomes available, that is to say every time portEnabled (which is equivalent to the MAC_Operational parameter for the ISS) becomes true permeates the "Port Access Control Protocol" specified in 802.1X-2004 Clause 8. The questions that this note poses are:

a) Can the state machines in 802.1X-2004 be readily adapted to cover the asynchronous acquisition of CAKs?
b) If they are adapted then how should the parts that are unwanted for MACsec (e.g. support for ControlledDirections) be handled?
c) If the existing state machines have to be changed considerably then is it better to do that with a fresh description, in a new clause, so as to minimize the risk of affecting existing .1X conformance claims?
d) If separate state machines are to be provided should there be two (or possibly even more) distinct classes of PAE?

It is the purpose of this note to try and provided insight into what the answers to these questions should be. It does not attempt to answer the question as to whether infrastructure authentication should be being provided by EAP in the first place, but simply assumes that is the case. In any event it would seem sensible to partition the task to be carried out by the MACsec key agreement (MKA) protocol operated by the SecY as starting with a CAK, that might be provided by some means other than EAP: assuming EAP will be used does not mean that the remainder of the .1af effort is useless if it is not.

2. Synchronous authentication

802.1X 2004 Clause 8 specifies that authentication is synchronized with availability of physical connectivity. This assumption runs very deep, even to the extent that (according to E.2.1) the Supplicant state machine expects the higher layer to watch this signal (portEnabled) "and begin an initialization process when the signal is asserted. This ensures that both the PAE state machine and the higher layer are in sync at initialization time. It is expected that the higher layer will reset both the eapSuccess and eapFail at this time." This is not best state machine practice, the only signal having the magic property of guaranteed initialization of all state machines being BEGIN. The current state machines all seem to be missing BEGIN, so it is difficult to tell when portEnabled is being used to substitute for BEGIN and when portEnabled is really meant. In any case this functionality is duplicated by eapRestart (E.2.2) which is used to communicate properly.

3. Current PAE to PAC coupling

This section (3) discusses some of the ways that 802.1X-2004 Clause 8 is coupled directly to the implicit model of access control as a set of switches used in .1X-2004. That coupling is a little less clear than it might be because no entity is been defined as implementing the switches, and some of the controls allude to the use of underlying session keys (e.g. those implemented by 802.11i) without being completely explicit about the layering model assumed. Prior notes and presentations have defined the "PAC" or Port Access Controller to implement the required controls, and that is destined in P802.1af-D0.2. To avoid any possible

†1 Ways of using EAP to support the eventual sharing of a CAK for a group CA have been discussed, and are not conceptually difficult, but the point-to-point case is very significant in its own right and requires resolution of some difficulties that apply to all cases, so there is little point in muddying the discussion.

†2 Throughout this note the term "CAK" is frequently used to refer both to a key and to data bound to or associated with that key (it's name for example).

ambiguity the rest of this note will describe 802.1X-2004 as if the PAC had been fully defined. In those terms it is apparent why 802.1X-2004 is called “PACP state machines”. They have been specified exactly to control the PAC, not to express a more general relationship between EAP and the range of possible entities that could secure data traffic. In that respect it is a bit of a mystery to me how some of the 2004 Clause 8 provisions can or are reconciled with 802.11i.

The controls that seem to depend on the very simple PAC include *portControl*. This variable is derived from *AuthControlledPortControl* and *SystemAuthControl*. If portControl is *ForceAuthorized* then the controlled Port is “held in the Authorized state”. It is very clear what this means for a simple PAC, but not at all clear what this could mean for an underlying service that requires to have a session key to function. There is no notion that there is a default session key, well known to all, or perhaps a default master key, similarly well known, that can be used to derive session keys in the absence of any real authorization. I suppose that it is just possible that such controls could be used to set MACsec management controls that are intended for incremental deployment, and allow the transmission of frames from the ControlledPort without protection. However the 2004 Clause 8 controls fall short of the provisions needed for incremental deployment themselves, and can only be translated in a clumsy way. Again I am not sure what 802.11i makes of these controls, and where that is documented.

One way of proceeding that seems very attractive is to remove the *AuthControlledPortControl* and *SystemAuthControl* administrative controls from the PAE and make them part of the PAC. For .1af this means moving them from 6.5.1 (where they are in D0.2) to a subclause of 6.4. This is consistent with our discussion of *OperControlledDirections* (currently in 6.4.2) in the July meeting. Again it is hard to reconcile the idea that authorization is only to be applied to one direction of communication with a underlying service that uses encryption.

If the 2004 Clause 8 state machines are to remain PAC specific then all the above mentioned controls could stay in the unmodified clause, whether the variables are thought to be part of the “PAE for the PAC” or are part of the PAC itself. If the Clause 8 specification is to be truly generalized they should be moved.

4. Generalized PAE state machines

This section (4) discusses what a set of PAE state machines that were general enough to cover easily host connect with a physically secure LAN (with a PAC), host connect using MACsec, a point-to-point infrastructure support using MACsec might look like. To simplify the discussion only the Supplicant will be considered. The Port Timers transmit machine is functionally trivial, so that leaves the following 2004 Clause 8 state machines for consideration:

a) Supplicant Key Transmit

b) Supplicant PAE

c) Supplicant Backend

d) Key Receive

I am entirely unclear why an EAPOL-Key message should be used for the purposes described in .1X-2004, but a little research leads me to believe that the EAPOL-Key message is now treated as if it carried an arbitrary payload, and its use is perhaps not dependent upon using encryption already setup between Supplicant and Authenticator as described in 2004 clause 8.1.9. For example the first .11 EAPOL-Key message that contains “Nonce 1” and “MAC 1” is not protected. It seems that Figure 8-8 in 8.1.9 should be ignored, and therefore the two supplicant key machines amount to nothing more than “receive the contents of an EAPOL-Key message if one is sent to you, and transmit an EAPOL-Key message if you have the data to do that and want to”. In that case it seems that the EAOPOL-Key messages are nothing more than a possible protocol identification method to carry key agreement protocol.

The relationship between the Supplicant PAE and Supplicant Backend state machine is not explained in .1X-2004 (as far as I can see), other than as a consequence of the detailed description of variables, so a short description is provided here. The whole of the Supplicant Backend machine is, with the minor exception of resource initialization which could be carried out anywhere, an expansion of the first part of the AUTHENTICATING state of the Supplicant PAE state machine. If the authentication within the backend machine succeeds the keyRun variable is set to prompt installation of the transmit key. The AUTHENTICATING state is only left after the key has been installed and portValid is set. The AUTHENTICATED state is then entered until receipt of another EAP messages restarts the authentication dialogue, or until portValid becomes false.

Taken literally the backend machine imposes restrictions on the EAP dialogue conducted by the Supplicant: it can transmit only in response to received EAP messages, and at most one message can be sent in response to each message received. Note that there is one exception to this rule, the procedure txStart() transmits an EAPOL-Start frame in the Supplicant PAE CONNECTING state. However this provision doesn’t allow the Supplicant to initiate the communication necessary to acquire a fresh key if it is already AUTHENTICATED.

In the usual application of EAP it is assumed that there is connectivity between the Authenticator and a backend Authentication Server, and that there is not necessarily connectivity between Supplicant and that server, except through the Authenticator. This assumption is probably best retained in the infrastructure application, so swapping roles from Supplicant to Authenticator is not a good way to overcome the restriction of a Supplicant not being able to acquire a fresh master key while authenticated. Indeed a more sweeping revision of the assumption that the underlying security can only make use of one key at a time is required if MACsec is to provide continuous connectivity with overlapping SAs. The next subclause (4.1) considers a repartitioning of the .1X machines to achieve this and other goals.

4.1 Repartitioned supplicant state machines

A slightly more general approach to supplicant state machines might proceed along the following lines.

At any time, when connectivity is available, a Supplicant might want to acquire a fresh or an additional master key. When the Supplicant has no master key that is of course an imperative, a necessary precondition to subsequent communication. However there are other reasons why a fresh master key may be wanted. A local policy rule might dictate that the existing master key is nearing the end of a lifetime set by policy, for example. The current state machine behavior can be seen as an example of a policy rule

that discards the current master key(s) whenever the physical connection is lost, forcing the acquisition of a new master key when the port is enabled once more.

Let's look at the heart of a machine or set of machines that acquire a new master key. For simplicity it is probably the case that only one master key should be being acquired at any one time, or at least only one so far as the role of Supplicant is being concerned. Assume the following to get started:

a) sAcquireMasterKey is a variable that signals to these machines to get them to run
b) the acquisition attempt is aborted if portEnabled transitions false while it is in progress (if portEnabled becomes true and a master key is still wanted then the acquisition attempt will naturally be restarted)
c) a quite separate machine is responsible for attempting to use the master key, to see if it is useful, has been installed successfully by the authenticator, etc.

Figure 1-1 is a state machine designed using .1X 2004 figures 8-17 (Supplicant PAE state machine) and 8-19 (Supplicant Backend state machine) as a starting point. It attempts to preserve some aspects of those machines that I don't fully understand, as well as fixing some apparent errors (these are described below in 4.2) while doing the basic job of those two machines. There are some differences in the expectations of the two machines. Basically Figure 1-1 is expected to run until eapSuccess becomes true, and then another machine is expected to collect the data (including a master key) associated with that data and then set eapSuccess false to indicate the data has been collected at the machine can proceed. Just as with the .1X 2004 originals the machine will support and authentication exchange initiated by the Authenticator, but has been modified to only stimulate authentication if a master key is wanted (the variable sAcquireMasterKey should be cleared before resetting eapSuccess, so that the machine doesn't initiate acquisition of a further key).

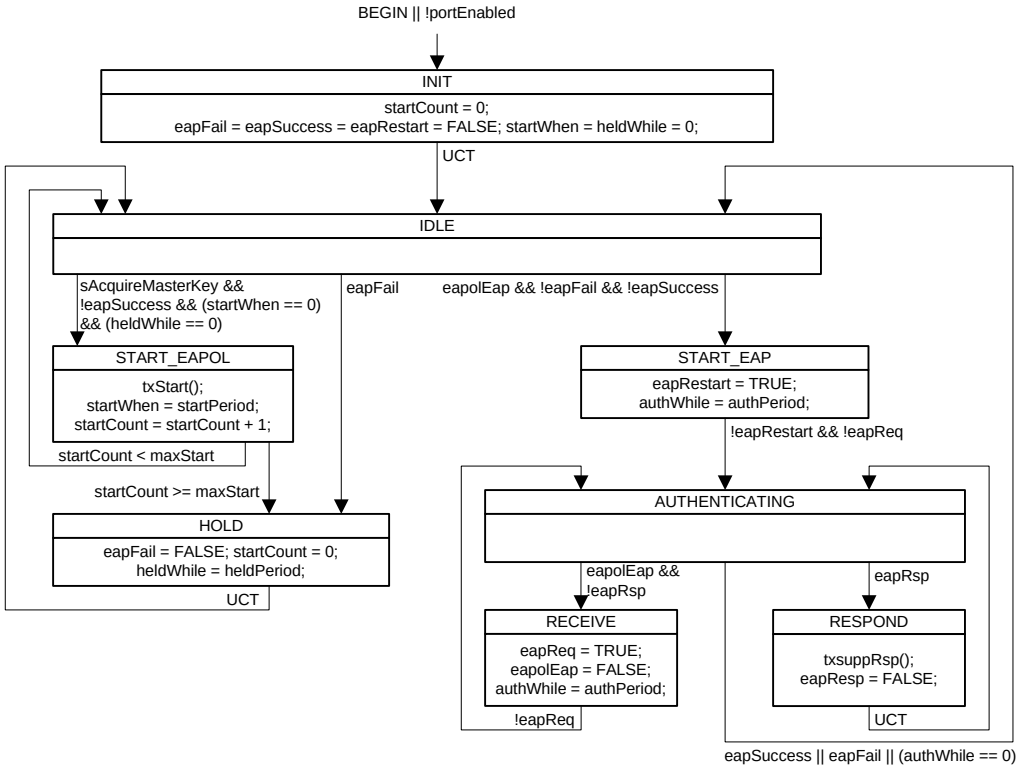

Figure 1-1—Supplicant state machine

4.2 Commentary on the Supplicant machines

In the course of examining the suitability of the existing .1X 2004 machines for use in the MACsec infrastructure application, and in developing Figure 1-1 I removed the following:

a) S_FORCE_AUTH, S_FORCE_UNAUTH states and all references to suppPortStatus, sPortMode, Force Authorized etc. All these are tied to the particular PAC (physically secure LAN application).
b) All references to Logoff, the point being that Logoff is just one way/policy of deciding that a key is no longer to be used. In the infrastructure application it would belong in a different state machine. There is some discussion necessary if logoff is not itself a protected message, and what the security exposure is of relying on it too heavily, as of course it may be intercepted.

I changed the following:

c) Since a prior master key may be still be in use, generating SAKs, and there is no wish to introduce a window of vulnerability as a new master key is acquired, Figure 1-1 does not attempt to install and check out the key before pronouncing success. That is left to another machine. If the key doesn't work then sAcquireMasterKey will be set once more to prompt acquisition of a further key. Hence there are no references to portValid.

I noticed a number of things, some of which I think are bugs in .1X-2004, or at least flapping loose ends:

d) The philosophy of using the eapRestart, eapReq, eapResp, eapNoResp, eapFail, eapSuccess variables is not consistently followed. It would seem that in some cases the variable is to be set by the machine sending the signal and the other machine clears it to accept the signal. However that means that there is no need for eapNoResp. To pace the delivery of EAP frames to the upper layer all that needs to do is delay setting eapReq.

e) Likewise I don't see why the procedure getSuppRsp() is defined, the upper layer will see eapReq and process it, and set eapResp to respond, so there is no need for the procedure call. The delay before the upper layer resets the eapReq is under its control, so doesn't need covering by a note about the procedure.

f) authWhile is inconsistently set in the backend machine.

g) eapFail is never reset.

h) eapolEap is inconsistently cleared in the backend machine.

i) There appear to be some state machine hazards as variables are assumed unchanged between states.

While the partitioning between the supplicant and its backend machine is a nice idea it doesn't seem to have worked out well. There are a number of controls whose purpose seems to be to limit the operation of the backend machine to inside the PAE machine AUTHENTICATING state. Some of the transitions into the AUTHENTICATING state (like eapFail) just rush through that machine with a change of variable name and fall straight out again. Putting the two machines together removed the need for the following:

j) suppAbort, suppStart, abortSupp().

k) suppFail, suppTimeout, suppSuccess.

There are some curiosities about the existing machines:

l) there is no mention of how eapSuccess or eapFail could be set in the CONNECTING state even before any EAP packet has been received. I hope this was not a "patch based on experience" that indicates that poor choices for timeout values have been deployed, but rather that it is possible for EAP to be configured with keys out of band or otherwise authorized.

5. Conclusions

It is still rather early to draw conclusions from the study so far. My preliminary thoughts are as follows.

a) The existing state (.1X-2004) machines are very tied to the PAC and its particular operation. A new set of machines are desirable for the infrastructure application.

b) The existing machines should be retained for the PAC application, unless there is overwhelming evidence that a new set of machines is easier to handle in that application.

c) On the other hand the communications philosophy and the variables and procedures already defined, though a little ragged in parts, would seem to be very useful without redefinition even with a new set of machines. Some variables may not be wanted, and a small number of additional variables may be needed but the existing set looks good.

These conclusions clearly need testing with other machines.