SCP2: Linkage Values

Summary

To support efficient revocation, end-entity certificates contain a linkage value (LV), which is derived from (cryptographic) linkage seed material. Publication of the seed is sufficient to revoke all certificates belonging to the revoked device, but without the seed an eavesdropper cannot tell which certificates belong to a particular device. Note: The revocation process is designed such that it does not give up backward privacy. For protection against insider attacks by the SCMS, the LV is the combination of two pre-linkage values (PLVs) produced by two independent LAs; this ensures that no single SCMS entity knows all the information belonging to a single device. An extension to the linkage values approach allows for group revocation, so that if all devices of a particular type have a flaw they can be revoked with a single entry on the revocation list, while keeping group membership secret until the relevant group seed is revealed. Group revocation is currently not implemented as no practical real-world use case been identified so far.

LVs and LAs are used to enable the SCMS to help achieve the following preliminary design requirements, which were developed by the research team to ensure appropriate privacy protections and efficiency:

1. There is an efficient way of revoking all the certificates within a device
2. Certificates are not linkable by an eavesdropper unless the owner has been revoked
3. A vehicle is trackable only after its credentials are revoked but not before it was revoked.
4. No single entity within the system is able to determine that two certificates belong to the same device. An exception to this rule is the Misbehavior Authority (MA).
5. No single entity within the SCMS is able to track a vehicle. Once a single LA is introduced, this requirement is not fulfilled any longer. For that reason, two LAs are used and the information which allows for tracking is split between them.

Description

The basic concept of LVs uses the well-known cryptographic construction known as a hash chain. As described above, a hash algorithm is like a cryptographic checksum; if the hash of ‘a’ is computed as H(a) = b, it is very hard for someone who sees only b to derive the input a, but given a and b it is trivial to determine that a hashes to b. Hence, it is desirable to have a series of identifiers in each certificate such that if a secret is revealed, the identifiers can be linked.

First a description of the revocation of individual nodes is provided. For simplicity, a system with a single LA that generates LVs is initially described. This system meets requirements 1), 2), and 3) discussed, above. It does not meet requirement 4), because the LA can link certificates. The following describes the basic process for a single series of certificates. A more detailed description will be provided below. For a complete description of the process see Section 4.2.2.

- LA starts with an initial linkage seed (ILS), ls(0). (This will be different for each vehicle.)
- For each time period i > 0, LA sets the LS ls(i) = H(ls(i-1)), for some hash function H (SHA-256, a National Institute for Standards and Technology (NIST)-approved standardized hash algorithm that is used throughout IEEE 1609.2 is employed)
- The certificate for each time period i contains the linkage value lv(i) = AES(ls(i), 0)
- To revoke a vehicle from time period i onwards, the revocation authority publishes ls(i)
- To check revocation at time period i’ > i, the recipient of a signed message:
    - Hashes ls(i), and then hashes the output of the hash, repeated (i’-i) times to obtain ls(i’)
    - Calculates lv(i’) = AES(ls(i'), 0)
    - Checks whether the certificate that signed the message contains the LV lv(i’). If it does, the certificate is considered revoked and the message is rejected

This achieves requirements 1), 2), and 3) as follows:

- Efficient revocation: Only one value needs be published to revoke all the certificates on a vehicle. The cost of maintaining the revocation data on the receiver side is one hash per revoked vehicle per time period. Hashing is very efficient, so this maintenance is inexpensive in terms of processing.
- Unlinkability against eavesdroppers: To tell if two certificates belong to the same vehicle, an eavesdropper would have to determine the two LSs ls1, ls2 that encrypt 0 to the PLVs plv1, plv2 in the certificates. Since AES is assumed to be a secure block cipher, this is not possible.
- Retrospective unlinkability: The hash chain can be run forward from the revocation value ls(i), but not backwards to recover previous values of ls (i). (This is a result of the non-invertibility of hash functions.)

However, the system has the problem that the LVs are generated centrally and the entity that generates the LVs knows the complete set of values that belong to a vehicle. To overcome this problem, the CAMP SCMS uses two LAs: LA1 and LA2.

In the description above, there is a single chain of LSs and LVs. In the CAMP SCMS, each of the LAs generates a chain of PLVs. These PLVs are individually encrypted and passed to the PCA; the PCA then XORs them together to obtain the LV that is put in the certificate. Now neither of the LAs knows the XORed linkage values that appear in the final certificate, because neither knows the values produced by the other LA. To revoke, the MA publishes the LSs from both LAs, and the recipient reconstructs both chains of PLVs and carries out the XORing to obtain the LVs for revoked certificates. The following describes the generation process in more detail:

- LA1 starts with a random ILS, ls1(0)
- LA2 starts with a random ILS, ls2(0)
- For each time period i > 0:
    - LA1 sets its LS ls1(i) = H(ls1(i-1)), and LA2 sets its LS ls2(i) = H(ls2(i-1))
    - LA1 sets its PLV, defined as plv1(i) = AES(ls1(i), 0) and LA2 sets its plv2(i) = AES(ls2(i), 0)
    - The CA sets the LV lv(i) = plv1(i) XOR plv2(i) and puts it in the certificate for time period i
- To revoke a vehicle from time period i onward, the revocation authority publishes the linkage seeds ls1(i) and ls2(i)
- To check revocation at time period i’ > i, the recipient of a signed message:
    - Iteratively hashes ls1(i) (i’-i) times to obtain ls1(i’); does the same for ls2(i)
    - Calculates PLVs plv1(i’) = AES(ls1(i'), 0) and plv2(i’) = AES(ls2(i'), 0)

- Checks whether the certificate that signed the message contains the LV lv(i') = plv1(i') XOR plv2(i'). If it does, the certificate is considered revoked and the message is rejected.

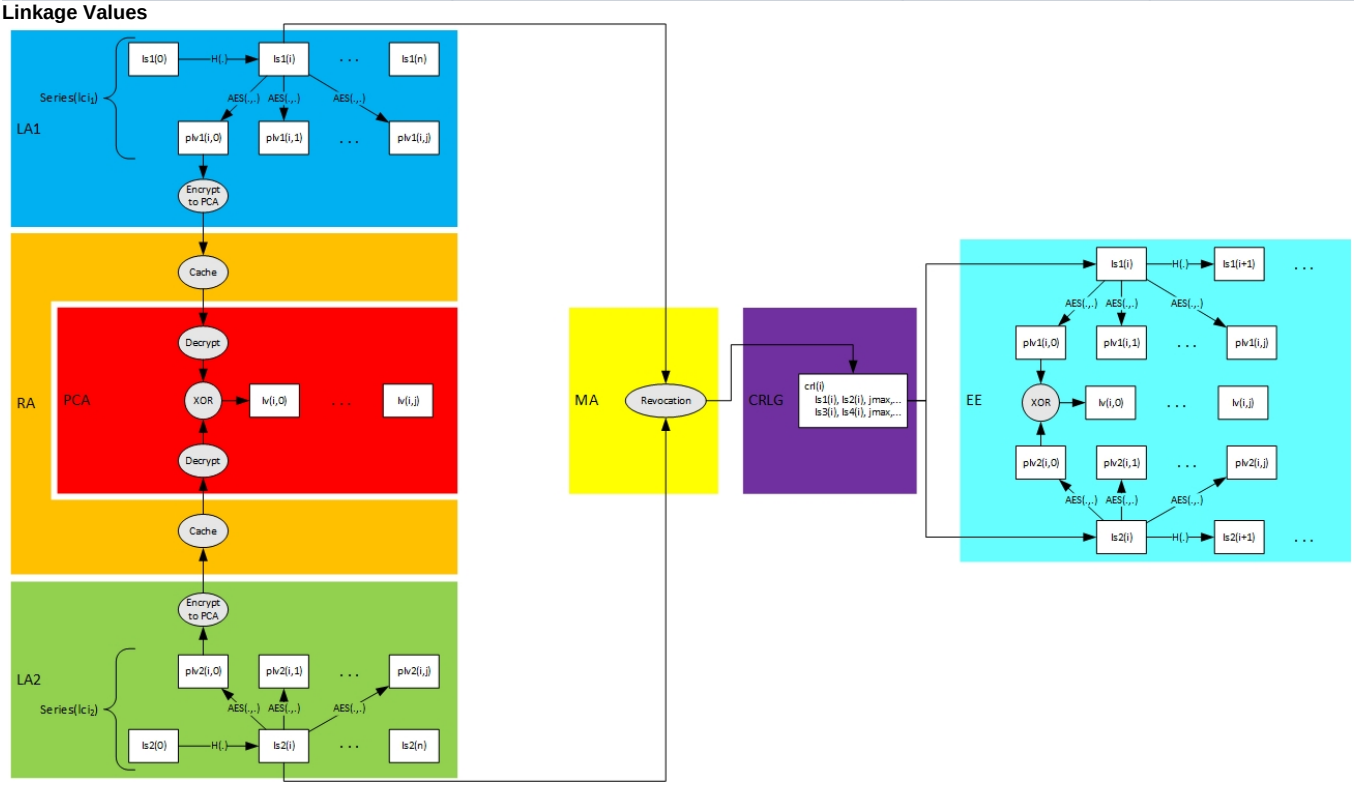
Four additional refinements in the CAMP SCMS are:

1. Instead of using a single time period counter i, time periods are denoted (i, j), where i counts up larger time periods (e.g., a day, a week, etc.) and j can be used in one of (at least) two ways: (a) for non-overlapping certificates, it can count up smaller time intervals within the larger time periods (e.g., 5-minute intervals); (b) for overlapping certificates, it can specify the number of certificates that are valid in a given time period i (e.g., fixing the range of j as 1-20 would imply that 20 certificates are valid simultaneously). The LSs ls1(i) and ls2(i) are calculated as described above, but the PLVs plv1(i, j) and plv2(i, j) are calculated as AES(ls1(i), j) and AES(ls2(i), j) , respectively. The reason for this is to save time for vehicles that have been offline for some time. If a vehicle has been turned off for 1 year, without this refinement, at key-on the vehicle will have to carry out 52 * 20 hashes for each revocation entry to bring its revocation information up to date (assuming that a vehicle is issued 20 simultaneously-valid certificates per week). With this refinement, the vehicle only has to perform one hash per week for each revocation entry. If revocation lists get large, this efficiency gain may be very useful.
2. To reduce the chance of collisions in the PLVs between two LAs, their identities are also employed during the computation of LSs and PLVs: la_id1 and la_id2 are unique 16-bit identity strings associated with LA1 and LA2, respectively. The LSs are calculated as: ls1(i) = H(la_id1 || ls1(i-1)), ls2(i) = H(la_id2 || ls2(i-1)). The PLVs are calculated as: plv1(i,j) = AES(ls1(i), la_id1 || j), plv2(i,j) = AES(ls2(i), la_id2 || j). This means that even if two LAs produce the same LS for a given time period, they will produce different sets of PLVs (because of the use of the identifier to produce the PLV from the LS), and their LSs will be different in the next time period (because of the use of the identifier to create the next seed from the current seed).
3. To reduce the size of certificate revocation list (CRL), which contains the LSs of the revoked vehicles, the LSs are truncated to 16 bytes.
4. Instead of plain AES, AES is used in the Davies-Meyer mode as a derivation function, which is basically XORing the output of AES with the input. In particular, for a key k and message m, instead of $AES_k(m)$, $(AES_k(m)$ XOR m) is returned for every invocation of AES.

The table below summarizes the linkage value generation and the figure below visualizes the described scheme. Test vectors for Linkage Values are available at http://stash.campllc.org/projects/SCMS/repos/crypto-test-vectors/browse/lv.txt

LA1	LA2	RA	PCA
1. Generate initial linkage seed, ls1(0) (128-bit string chosen at random for every device). 2. Compute linkage seed for i^{th} period through an iterative process defined as: ls1(i) = [SHA-256(la_id1 || ls1(i-1) || 0^{112})]$_{128}$. 3. Compute pre-linkage value, plv1(i, j) = [(AES(ls1(i), la_id1 || j || 0^{80})) XOR (la_id1 || j || 0^{80})]$_{72}$. 4. Encrypt plv1(i, j) for PCA, and send it to RA.	1. Generate initial linkage seed, ls2(0) (128-bit string chosen at random for every device). 2. Compute linkage seed for i^{th} period through an iterative process defined as: ls2(i) = [SHA-256(la_id2 || ls2(i-1) || 0^{112})]$_{128}$. 3. Compute pre-linkage value, plv2(i, j) = [(AES(ls2(i), la_id2 || j || 0^{80})) XOR (la_id2 || j || 0^{80})]$_{72}$. 4. Encrypt plv2(i, j) for PCA, and send it to RA.	5. Include encrypted plv1(i, j) and plv2(i, j) in the certificate request.	6. Decrypt the packet from RA to obtain plv1(i, j) and plv2(i, j). 7. Compute linkage value, lv(i, j) = plv1(i, j) XOR plv2(i, j)

Linkage Values



Creation of Individual Linkage Values and Revocation of Individual Device