Assignment #5 – Solutions

Problem 1

- Suppose that ORCA cards and readers are only capable of symmetric (e.g. AES) encryption but you want each card to have its own symmetric key for communications with readers. Although you don't like it, you are assured that there is enough physical security around the readers so that they can all share a single key.

  Describe a design wherein each card can use its unique symmetric key to communicate securely with any reader.

Problem 1

- Readers share secret symmetric key k.
- Cards each have a unique symmetric key k_i

- A couple possible solutions:

1. On card i, store k_i and $E_k(k_i)$. When communicating with a reader, send $E_k(k_i)$ from the card to the reader. The reader can decrypt $E_k(k_i)$, retrieving k_i. Then the card & reader can communicate using k_i.
    - This is like pre-loading a Kerberos ticket for the reader "service" onto each card.

Problem 1

2. On card i, store i and k_i, where $k_i = \text{HMAC}(k, i)$. When communicating with a reader, send i from the card to the reader. The reader can then derive k_i from k and i. Then the card & reader can communicate using k_i.

Problem 2

- "Fun with CRLs"
- 1,000,000 ORCA cards, 1%/month loss rate.
- Q: How big is the CRL in the steady state if you have to hold 2 years of info?
    - Assume CRL requires 512 bytes of fixed information plus 36 bytes of storage per revocation entry when ASN.1 encoded.

- 1,000,000 cards, 1% month → 24% loss over 2 years
  → 240,000 entries on the CRL

- CRL size: (240,000 * 36) + 512 bytes = 8,640,512 bytes

Problem 3

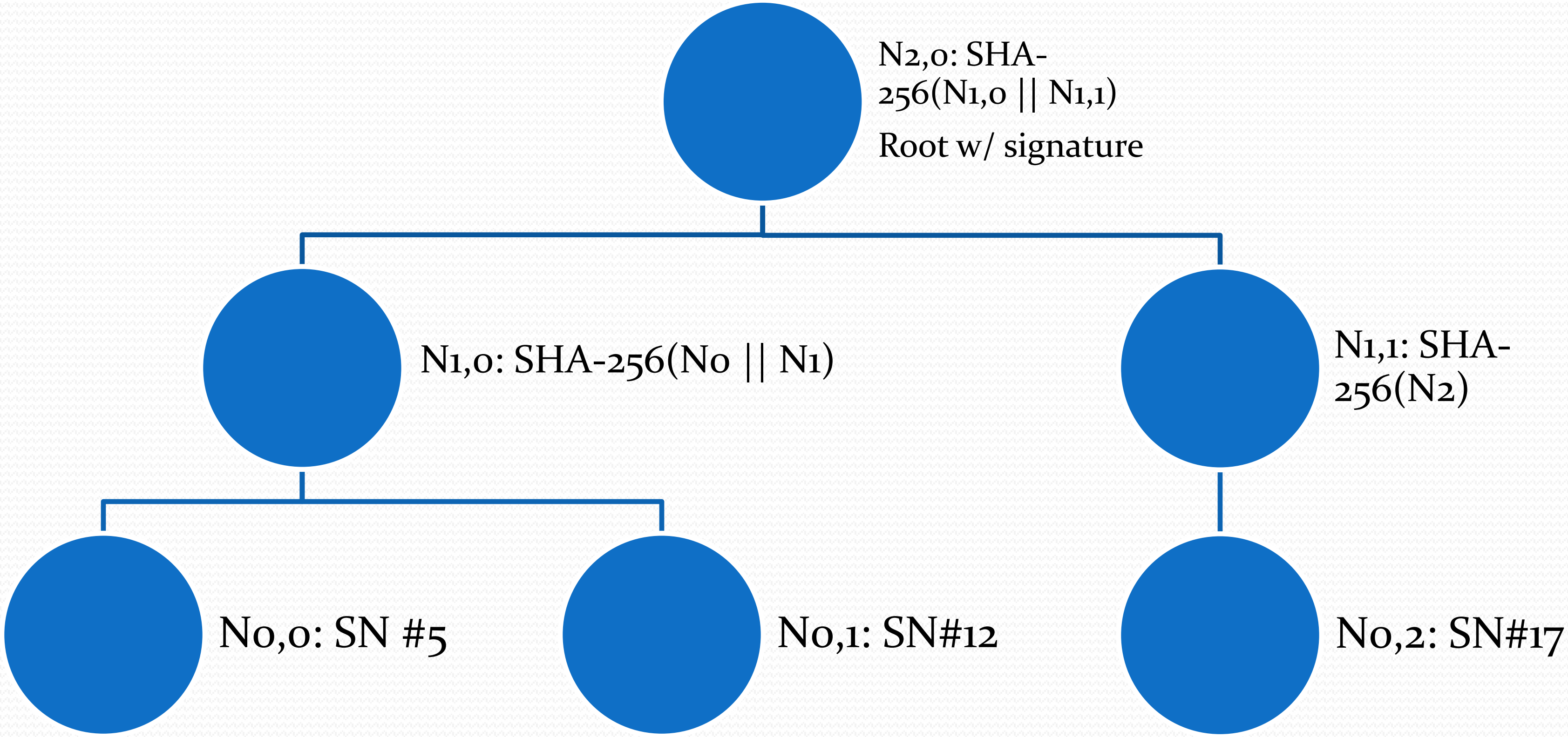
- "Fun with CRLs, Part II"
- Design an alternative data structure for holding CRL information on each reader that has the following properties, where m is the total number of revoked cards in the data structure and n is the number of additions and deletions made to the data structure in a single day:
  - Each day's incremental updates involve only $O(n \ \log m)$ modifications to the data structure.
  - Searching the "CRL" for an entry takes $O(\log m)$ time.
- Like a "regular" CRL, the data structure is always integrity-protected with a digital signature from the CA.

Problem 3

- Solution: “Certificate Revocation Trees”*
- Basic idea: Create a tree data structure where the leaves of the tree hold (in sorted order) the serial numbers of revoked certs.
  - Intermediate parent nodes are formed by hashing the contents of the node’s children.
  - Digitally sign the root node using the CA’s private key and distribute that signature as part of the root node.

*Kocher, “On Certificate Revocation and Validation,” Proceedings of Financial Cryptography ‘98, LCS 1465, pp. 127-177.

Problem 3


N2,0: SHA-256(N1,0 || N1,1)
Root w/ signature
N1,0: SHA-256(N0 || N1)
N1,1: SHA-256(N2)
N0,0: SN #5
N0,1: SN#12
N0,2: SN#17

Problem 3

- Updates:
  - Just need to update leaf nodes that change on each incremental update and the intermediate nodes between each changed leaf and the root.
  - Also need to send a new signature on the root each time (because any change will change the root).

Problem 4

- Pres and two VPs share modulus $N = pq$.
- Pres uses public exponent $e = 65537$.
- VPs use public exponents $e_1 = 3$ and $e_2 = 5$.

$$m^{15d_1d_2} \bmod N = \left(m^{3d_1}\right)^{5d_2} \bmod N = m^{5d_2}\ mod\ N = m$$

The bad news: A corresponding (d, e) pair allows you to factor N.

Problem 4 (cont.)

What can you do with the VP keys?

$$de = k(p-1)(q-1)$$
$$3d_1 = k_1(p-1)(q-1)$$
$$5d_2 = k_2(p-1)(q-1)$$

k_1 and k_2 must be *very* small. Given one of d_1 and d_2, one need try very few possibilities for k_1 and k_2 to derive the d_i that you *don't* already have.