CS5430 Homework 3: Public Key Infrastructure

General Instructions. You are expected to work alone on this assignment.

Due: October 14, 2022 11:59pm. No late assignments will be accepted.

Submit your solution using CMS. Prepare your solution as .pdf, as follows:

- Use 10 point or larger font.
- Submit each problem (as a separate file) into the correct CMS submission box for that problem.

Problem 1: Threshold Digital Signatures.

Given a secret bit string *s*, define an *n*-way split of s to be a set of *n* shares $s_1, s_2, \ldots, s_n$ such that
- SS1: Secret *s* can be reconstructed from the *n* shares.
- SS2: Nothing about *s* can be inferred from any subset of $n-1$ or fewer shares.

The following is proposed as an implementation of an *n*-way split for a secret *s*, where $0 \le s \le p-1$ holds for a prime number p that is publicly known.
(i) For each share s_i where $1 \le i \le n-1$: choose a random integer between 0 and $p-1$.
(ii) Define s_n to be: $(s - \sum_{1 \le i < n} s_i) \bmod p$

(a) Give a procedure to reconstruct s from a set of n shares.

(b) Prove that a set of $n-1$ shares reveal no information about the secret s. Your proof should show that any value of s would be possible given any set of $n-1$ shares.

(c) A digital signature scheme provides two functions, assuming *k* is a private key and *K* is the corresponding public key.

k-Sign(*m*): uses private key *k* to produce a digital signature σ_m for *m*.

K-Verify (σ_m, m): returns $true$ if and only if σ_m is the digital signature generated for message *m* using private key *k* corresponding to public key *K*.

Suppose signatures are being produced by using exponentiation in *mod p*, as follows.

k-Sign(*m*): $(m^k \bmod p)$

And suppose *K*-Verify (σ_m, m) is defined appropriately.

Given a private key *k* that is a (secret) bit string, select a random value for each of $k_1, k_2, \ldots k_n$ that together satisfy

$$k = k_1 + k_2 + \cdots + k_n$$

Using these, we can construct a set of *n partial signatures* by invoking *k*-Sign(·) with each of the k_i:

k_i-Sign(*m*): $(m^{k_i} \bmod p)$

Can *k*-Sign(*m*) be recovered from the set of *n* partial signatures k_i-Sign(*m*) where $1 \leq i \leq n$? Show how or explain why this would not be possible.

Problem 2: Self-signed Certificates.

Many CA schemes employ a *self-signed certificate* for the start of a certificate chain. This is a certificate that is signed using a private key (e.g., k_{FBS}) where that private key can only be accessed by the principal (e.g. *FBS*) that the self-signed certificate is binding to the corresponding public key (e.g., K_{FBS}). Here is an example:

$\langle K_{FBS}$ **speaksfor** $FBS \rangle k_{FBS}$

(a) What is the formaization of this message in terms of **says** and **speaksfor** operators.

(b) What -- if anything -- useful can be inferred from the message?

Problem 3: Inferences about belief sets.

The logic for **says** and **speaksfor** operators includes an inference rule R3:

$$\frac{A \textbf{ speaksfor } B, \quad A \textbf{ says } S}{B \textbf{ says } S}$$

The hypotheses of the rule (i.e. "*A* **speaksfor** *B*" and "*A* **says** *S*") are statements about the sets of beliefs that *A* and *B* hold. The conclusion of the rule (i.e., "*B* **says** *S*") is a statement about the set beliefs that *B* holds. To show that the rule R3 is sound, it suffices to show that the meaning of the statement in the conclusion of the rule will be a true statement if the meanings of the statements in the hypotheses are also true statements. Use this approach to show that inference rule R3 is sound.

Problem 4: Certificate Transparency.

The implementation of certificate transparency involves a Merkle Hash Tree (MHT) that stores an append-only log of certificates that have been registered by some CA's. By periodically checking this append-only log, a service can see whether a bogus certificate for that service has been registered. And to check whether a given certificate D_i is in the log, the MHT is traversed from a leaf with value D_i to the root. If the computed hash for the root hash equals a signed value that some trusted party previously computed, then D_i is indeed in the MHT.

(a) If D_i is the 247th certificate to have been added to the append-only log, what is the approximate worst-case length of the path to be traversed for looking up an arbitrary leaf? What, if any, role does SCT (signed certificate timestamp) and/or MMD (maximum merge delay) play in achieving this worst case?

(b) Under what conditions, if any, is it permissible for the service that is storing MHT to construct a new Merkel Hash Tree that contains exactly the same leaves but is better balanced that the original?

(c) The standard certificate transparency protocol uses an append-only log and, therefore, the contents of that log are never deleted. Consider a variation of that scheme. In this variation, leaves in the MHT are allowed to be deleted. What restrictions would clients and servers have to satisfy for it to be permissible to delete D_i from the MHT?