CS5432 Homework 1: PUFs and TPMs

General Instructions. You may (but do not have to) collaborate with one other student on this assignment. If you do collaborate then both students should form a CMS group and submit their solution to that group. Both students are responsible for all of the answers.

Due: March 8, 2021 at 11:59pm. No late assignments will be accepted.

Submit your solution using CMS. Typeset your solution to produce .pdf, as follows:

- Use 10 point or larger font.
- Start each problem on a separate page.

Problem 1. Claude Shannon's *one-time pad* encryption uses exclusive-or operations ($\oplus$) to produce a ciphertext from
plaintext. Given a fresh n-bit random bit string k[1..n] serving as the key and an n-bit plaintext bit string p[1..n], then corresponding ciphertext c[1..n] would be computed according to

$$c[i] := k[i] \oplus p[i] \text{ for } 1 \leq i \leq n$$

The plaintext would be recovered from that ciphertext using another bit-wise exclusive or operation with the key:

$$p[i] := c[i] \oplus k[i] \text{ for } 1 \leq i \leq n,$$

Decryption of c recovers p because $\oplus$ is commutative, $x \oplus x = 0$ (for all x), and $x \oplus 0 = x$ (for all x).

It is crucial to security of this scheme that a given key be used for encrypting at most one message.

A proof. Here is a proof that ciphertext c is confidential. We show that any ciphertext could be encrypting any plaintext—a property known as *perfect secrecy*. Since any ciphertext could correspond to any plaintext, having the ciphertext (without the key) will reveal nothing about the plaintext it encrypts (which is equivalent to achieving confidentiality for the plaintext). The crucial observation is that given a ciphertext c[1..n] and a plaintext p[1..n] there always exists a bitstring b[1..n] such that

$$c[i] := b[i] \oplus p[i] \text{ for } 1 \leq i \leq n$$

So we constructed a bit string b[1..n] that could serve as the key to generate the given ciphertext from the given plaintext. That is, we have shown that ciphertext c could have been generated from any plaintext.

A question. Consider an IOT device that is intended to run once, for 24 hours. While operating, it outputs a steady stream of bits, representing s-bit sensor values that are measured each second. The stream is encrypted using a one-time pad k[1..] for encrypting and later decrypting the message. A PUF advocate is considering two ways that one-time pad might be generated by using a weak PUF implementing a function F(x) that produces an unpredictable m-bit output for each input. The IOT device also contains an implementation of a well known hash function H(x) the produces m-bit strings.

(a) Use F(0) for the first m bits of k, use F(1) for the next m bits, and so on.

(b) Use F(0) for the first m bits of k, use H(F(0) for the next m bits, use Hash(Hash(F(0))) for the next m bits, and so on.

Comment on the practicality, feasibility, and security of each approach.

Problem 2. Here is a definition of a naming function N(·) for measured principals. It uses concatenation (**concat**) of bit strings rather than computing successive hashes

N(d0 d1 … dn) = Hash(d0) **concat** Hash(d1) … **concat** Hash(dn)

N(·) is efficient to compute and can be computed incrementally, just like definition (11.1) of N(·) in the chapter 11 reading. Is using N(·) for names of principals as secure as the security that definition (11.1) provides? If not, explain what properties the new definition lacks and give an attack.

Problem 3. The **QKRgen** instruction (page 311, in the chapter 11 reading) generates a digitally signed bit string that includes a prefix "qkr key:"

(a) What is the justification the reading gives for including such prefixes in digital signatures produced by executing TPM instructions?

(b) Is such a prefix actually necessary for the digital signatures generated by the **QKRgen**?

Problem 4. Names of measured principals are calculated from sequences of descriptors. There are several possible ways for computing a descriptor for the information being stored in the heap of a UNIX process that uses the heap **only** for storing a single linked list, constructed from **type** node, defined by:

type node = **record** integer : data ; ptr to node : next **end**

Here are three proposals.

(i) Compute the hash of the entire memory region that is available for use by data structures being stored in the heap.

(ii) Traverse the linked list being stored in the heap by the process, incorporating a hash of each entire node (data and next) in the list, as it is reached. That is, the hash LH for the list is updated, as follows, as each node N in the linked list is reached:

LH := Hash(LH, N)

(iii) Traverse the linked list in the heap, incorporating a hash of only the data part of each node, as that node is reached. That is, the hash LH for the list is updated, as follows, as each node N is reached:

LH := Hash(LH, N.data)

Discuss the advantages and disadvantages of each scheme for use in creating the name of a measured principal, where that name will be used to control access to a key that stores data between sessions. Assume each session begins with some computation that builds the list in the heap anew, where that list contains exactly the same sequence of data values as it did in the previous session.

.