CS 355: Topics in Cryptography **Spring 2021**

Problem Set 1

Due: April 12, 2021 at 11:59pm PT (submit via Gradescope)

Instructions: You **must** typeset your solution in LaTeX using the provided template:

https://crypto.stanford.edu/cs355/21sp/homework.tex

Submission Instructions: You must submit your problem set via Gradescope. Please use course code **V8WVZK** to sign up. Note that Gradescope requires that the solution to each problem starts on a **new page**.

Bugs: We make mistakes! If it looks like there might be a mistake in the statement of a problem, please ask a clarifying question on Piazza.

Problem 1: Conceptual Questions [8 points]. For each of the following statements, say whether it is TRUE or FALSE. Write *at most one sentence* to justify your answer.

(a) Which of the following are true in a world where $\mathsf{P} = \mathsf{NP}$.

i Secure PRFs exist in the standard model.

ii Secure PRFs exist in the random oracle model.

iii The one-time-pad cipher is secure.

(b) If there exists a PRG with 1-bit stretch, there exists a PRG with n^{800}-bit stretch (where n is the length of the PRG seed).

(c) Let $P\colon \mathcal{K} \times \mathcal{X} \to \mathcal{X}$ be a pseudorandom permutation. Then:

i $f_{k=0}(x) := P(0,x)$ is (always) a one way function.

ii $f_{k=0}(x) := P(0,x)$ is (always) a one way permutation.

iii $f_{x=0}(k) := P(k,0)$ is (always) a one way function.

iv $f_{x=0}(k) := P(k,0)$ is (always) a one way permutation.

Problem 2: Key Leakage in PRFs [5 points]. Let F be a secure PRF defined over $(\mathcal{K},\mathcal{X},\mathcal{Y})$, where $\mathcal{K} = \mathcal{X} = \mathcal{Y} = \{0,1\}^n$. Let $\mathcal{K}_1 = \{0,1\}^{n+1}$. Construct a new PRF F_1, defined over $(\mathcal{K}_1,\mathcal{X},\mathcal{Y})$, with the following property: the PRF F_1 is secure; however, if the adversary learns the last bit of the key then the PRF is no longer secure. This shows that leaking even a *single* bit of the secret key can completely destroy the PRF security property.
[**Hint:** Try changing the value of F at a single point.]

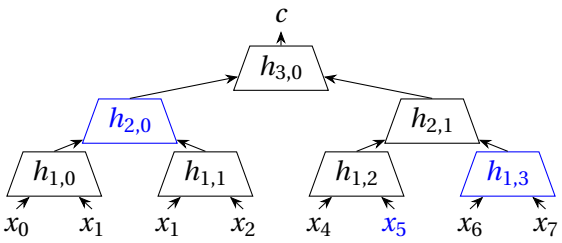

Figure 1: A Merkle tree

Problem 3: Multi-Commitments [10 points]. Let $\mathbb{G}$ be a group of prime order q in which the discrete logarithm problem is hard. Let g and h be generators of $\mathbb{G}$. As we saw in class, the Pedersen commitment scheme commits to a message $m \in \mathbb{Z}_q$ using randomness $r \in \mathbb{Z}_q$ as $\mathsf{Commit}(m;r) := g^m h^r \in \mathbb{G}$. Moreover, we saw that Pedersen commitments are *additively homomorphic*, meaning that given commitments to m_1 and m_2, one can compute a commitment to $m_1 + m_2$. The "public parameters" associated with the Pedersen commitment scheme are the description of the prime-order group $\mathbb{G}$ and the group elements g and h.

(a) Use $\mathbb{G}$ to construct an additively homomorphic commitment scheme $\mathsf{Commit}_n(m_1,\ldots,m_n;r)$ that commits to a length-n vector of messages $(m_1,\ldots,m_n) \in \mathbb{Z}_q^n$ using randomness $r \in \mathbb{Z}_q$. The output of the commitment should be short – only a single group element. You should specify both the public parameters of your scheme (which may be different from that of the basic Pedersen commitment scheme) as well as the description of the Commit_n function.

(b) Prove that your commitment scheme is perfectly hiding and computationally binding (assuming hardness of discrete log in $\mathbb{G}$).

(c) Show that if you are given a hash function $H\colon \mathbb{Z}_q \to \mathbb{G}$ (modeled as a random oracle), the public parameters for your construction from Part (a) only needs to consist of the description of the group $\mathbb{G}$ and the description of H. Argue *informally* why your construction is secure. You do *not* need to provide a formal proof. This problem shows that getting rid of public parameters is another reason why random oracles are useful in practice!

(d) **Extra Credit [3 points].** Prove formally that your construction from Part (c) is secure in the random oracle model.

Problem 4: Vector Commitments [5 points]. The previous problem considered multi-commitments: commitments to vectors which can be opened all at once. In this problem we consider *vector commitments*: commitments to vectors which can be opened to one index at a time.

The classic construction of a vector commitment scheme is the *Merkle tree*, which is parameterized by a hash function $H : \mathcal{X} \times \mathcal{X} \to \mathcal{X}$. The core of this construction is a hash tree, as shown in Figure 1. Each internal node in the tree represents an evaluation of the hash function over the child nodes. The Commit procedure evaluates the hash tree, returning the root at the commitment. The IdxOpen procedure produces the siblings along a path (e.g., the blue nodes, for x_4), and the IdxVerify checks the hash evaluations along the path.

More precisely, the Merkle tree vector commitment scheme is defined by three algorithms:

- $\mathsf{Commit}(d \in \mathbb{N}, x \in \mathcal{X}^{2^d}) \to \mathcal{X}$:

  for $i \in \{0,1,\ldots,2^d-1\}$: $h_{0,i} \leftarrow x_i$

  for $\ell \in \{1,2,\ldots,d\}$, for $i \in \{0,1,\ldots,2^{d-\ell}-1\}$: $h_{\ell,i} \leftarrow H(h_{\ell-1,2i}, h_{\ell-1,2i+1})$

  return $h_{d,0}$

- $\mathsf{IdxOpen}(d \in \mathbb{N}, i \in \mathbb{N}, x \in \mathcal{X}^{2^d}) \to \mathcal{X}^d$:

  compute $h_{\ell,i}$ as in Commit

  for $\ell \in \{0,\ldots,d-1\}$: $p_i \leftarrow h_{\ell,((i \gg \ell) \oplus 1)}$[1]

  return $(p_0,\ldots,p_{d-1})$

- $\mathsf{IdxVerify}(d \in \mathbb{N}, i \in \mathbb{N}, x, \in \mathcal{X}, c \in \mathcal{X}, p \in \mathcal{X}^d) \to \{0,1\}$:

  $h_0 \leftarrow x$

  for $\ell \in \{1,\ldots,d\}$:

  $j \leftarrow i \gg (\ell-1)$

  if j is odd: $h_\ell \leftarrow H(p_{\ell-1}, h_{\ell-1})$

  else: $h_\ell \leftarrow H(h_{\ell-1}, p_{\ell-1})$

  return $h_d \stackrel{?}{=} c$

A vector commitment scheme is *index binding* if for all $d \in \mathbb{N}$ and for all efficient adversaries $\mathcal{A}$,

$$\Pr\left\{\mathsf{IdxVerify}(d,i,x,c,p) = 1 \wedge \mathsf{IdxVerify}(d,i,x',c,p') = 1 \wedge x \neq x' : (c,i,x,p,x',p') \leftarrow \mathcal{A}(d,\lambda)\right\} = \mathrm{negl}(\lambda)$$

where λ is the security parameter of H.

(a) Please prove that Merkle tree vector commitments are index binding if H is collision-resistant. Recall that a hash function H is *collision resistant* if for all efficient adversaries $\mathcal{A}$,

$$\Pr\left\{H(x,y) = H(x',y') \wedge (x,y) \neq (x',y') : (x,y,x',y') \leftarrow \mathcal{A}(\lambda)\right\} \leq \mathrm{negl}(\lambda)$$

(b) **Extra Credit [3 points].** Is a Merkle tree commitment hiding in the random oracle model? If so, prove it. If not, briefly explain why, modify the scheme to make it hiding in the random oracle model, and prove that the modification is hiding in the random oracle model. *Note: A few different "hiding" properties can be formalized for vector commitments. You should consider whether the commitment itself is hiding—you need not consider opening proofs.*

Problem 5: Our Favorite PRF [10 points]. In class we saw the PRF $F(k,x) = H(x)^k$ and were told that it has many useful properties. In this problem, we will explore two applications of this PRF.

[1] Here, $\gg$ is logical right shift and $\oplus$ is bitwise XOR, as in C. That is, $x \gg y$ denotes $\lfloor x/2^y \rfloor$ and $x \oplus y$ denotes the integer with binary representation equal to the bitwise XOR of the binary representations of x and y.

(a) Let $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ be a PRF defined over groups $(\mathcal{K}, +)$ and $(\mathcal{Y}, \otimes)$, where $+$ and $\otimes$ are the respective group operations in those groups. We say F is *key-homomorphic* if it holds that

$$F(k_1 + k_2, x) = F(k_1, x) \otimes F(k_2, x).$$

Is the PRF $F(k, x) = H(x)^k$ defined with a random oracle $H : \mathcal{X} \to G$ (where G is a group of prime order p) a key-homomorphic PRF? Please prove your answer one way or the other.

(b) *Key rotation* is a common problem encountered in cloud storage: how to change the key under which data is encrypted without sending the keys to the storage provider? A naive solution is to download the encrypted data, decrypt it, re-encrypt it under a new key, and re-upload the new ciphertext. We will now see how this process can be made more efficient with a key-homomorphic PRF.

Suppose you have a ciphertext c made up of blocks $c_1, ..., c_N$ that corresponds to a message $m = (m_1, ..., m_N)$ encrypted under a key k_1 using a key-homomorphic PRF F in counter mode, i.e., $c_i = m_i \otimes F(k_1, i)$. Now you want to rotate to a key k_2. It turns out you can send the storage provider a single element $k_{\mathsf{update}} \in \mathcal{K}$ which it can then use to generate c', an encryption of m under k_2. Please tell us how you can compute k_{update} and how the storage provider can use k_{update} and c to compute c'.

(c) An *oblivious PRF* is an interactive protocol between a client who holds a message x and a server who holds a key k. The protocol allows the client to learn the PRF evaluation $F(k, x)$ without the server learning anything about x. Oblivious PRFs are used in many advanced crypto protocols.

It turns out that there is an oblivious PRF protocol for the PRF $F(k, x) = H(x)^k$. Please show us how a client holding x and a server holding k can interact so that the client learns $H(x)^k$ while the server learns nothing about x. You don't need to prove security, we would just like to see the protocol.

Problem 6: Feedback [0 points]. Please answer the following questions to help us design future problem sets. You are not required to answer these questions, and if you would prefer to answer anonymously, please use this form. However, we do encourage you to provide us feedback on how to improve the course experience.

(a) Roughly how long did you spend on this problem set?

(b) What was your favorite problem on this problem set?

(c) What was your least favorite problem on this problem set?

(d) Any other feedback for this problem set?