CS 355: Topics in Cryptography **Spring 2026**

Problem Set 2

Due: Friday, 24 April 2026 (submit via Gradescope)

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

https://cs355.stanford.edu/homework.tex

Submission Instructions: You must submit your problem set via Gradescope. Please use course code **X85KN6** 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 Ed.

Note: The following two documents may help with number theory background on this assignment.

1. https://crypto.stanford.edu/~dabo/cs255/handouts/numth1.pdf
2. https://crypto.stanford.edu/~dabo/cs255/handouts/numth2.pdf

Problem 1: Random Oracle Commitments [5 points]. Let $H : \mathcal{M} \times R \to \mathcal{C}$ be a function where $\mathcal{R} = \mathcal{C} = \{0,1\}^\lambda$. Define $\mathsf{Commit}(m, r) = H(m, r)$. Show that Commit is binding and hiding if H is modeled as a random oracle.

Problem 2: Vector Commitments [5 points]. In this problem we consider *vector commitments*: commitments to vectors which can be opened to one index at a time.

The classic construction 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:

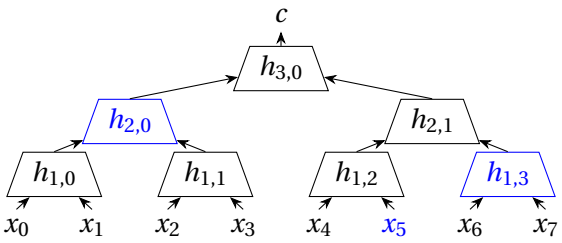

Figure 1: A Merkle tree

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

  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 $\mathcal{A}$,

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

where λ is the security parameter of H. A hash function H is *collision resistant* if for all efficient adversaries $\mathcal{A}$,

$$\Pr\left\{H(x,y) = H(x',y') \wedge (x,y) \neq (x',y') : (x,y,x',y') \leftarrow \mathcal{A}(1^\lambda)\right\} = \mathrm{negl}(\lambda)$$

(a) Please prove that Merkle tree vector commitments are index binding if H is collision-resistant.

(b) **Extra Credit [2 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 3: For each of the following statements, say whether it is TRUE or FALSE. Write *at most one sentence* to justify your answer [7 points].

1. Let p, q, r, and r' be distinct large primes. Let $N = pqr$ and $N' = pqr'$. Assume that there does *not* exist an efficient (probabilistic polynomial time) factoring algorithm. Say whether each of the following statements are TRUE or FALSE.

[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) There is an efficient algorithm that takes N as input and outputs r.

(b) There is an efficient algorithm that takes N and N' as input and outputs r.

(c) There is an efficient algorithm that takes N and N' as input and outputs q.

2. Let $\mathbb{G}$ be a group of prime order q. Consider the following special cases of the discrete-log problem. For each of them, say TRUE if an efficient (polynomial in $\log q$) algorithm for the special case can be used to construct an efficient algorithm for the general case of the discrete-log problem, and FALSE otherwise.

   (a) An algorithm that correctly outputs the discrete log only when it is smaller than $q/\log q$.

   (b) An algorithm that correctly outputs the discrete log only when it is smaller than $\log q$.

3. Given $g \in \mathbb{G}$ and a positive integer n, a generic group algorithm requires $\Omega(n)$ time to compute g^n.

4. Let $\mathbb{G}$ be a cyclic group of prime order q with a generator $g \in \mathbb{G}$ and $H\colon \mathbb{G} \to \{1,2,3\}$ be a random function. A walk on $\mathbb{G}$ defined as $x_0 \xleftarrow{R} \mathbb{G}$ and $x_{i+1} \leftarrow x_i \cdot g^{H(x_i)}$ collides in $O(\sqrt{q})$ steps in expectation (i.e., if $i_{\text{col}} = \min\{i \in \mathbb{N}\colon \exists j < i \text{ s.t. } x_i = x_j\}$, then $\mathbb{E}_{x_0,H}[i_{\text{col}}] \le O(\sqrt{q})$).

Problem 4: Generic Discrete Log [12 points]. You will need the "Birthday Bound" for these problems (see Boneh-Shoup, Appendix B.1). Let $\mathbb{G} = \langle g \rangle$ be a group of prime order q.

(a) Many times this quarter, we have mentioned that there is a algorithm for discrete log that runs in time $\tilde{O}(\sqrt{q})$.[2] This algorithm, which is shockingly simple, is the best known algorithm for discrete-log in standard elliptic curve groups.

You are given a discrete-log challenge $h = g^x \in \mathbb{G}$. Show that by computing two tables of values: one of the form g^{r_i} for $r_i \xleftarrow{R} \mathbb{Z}_q$ and one of the form h^{s_j} for $s_j \xleftarrow{R} \mathbb{Z}_q$, you can recover the discrete log $x \in \mathbb{Z}_q$ in time $\tilde{O}(\sqrt{q})$.

This algorithm uses space $\tilde{\Omega}(\sqrt{q})$. A clever trick, due to Pollard, gets the space usage down to $O(\text{poly}\log q)$.

(b) The B-bounded discrete-log problem is the problem of recovering $x \in \{0,\ldots,B\}$, given $g^x \in \mathbb{G}$. That is, the group $\mathbb{G}$ is of order q, but we sample the exponent x from a range of size $B \ll q$. Modify your algorithm of part (a) to solve the B-bounded discrete-log problem in time $\tilde{O}(\sqrt{B})$.

Problem 5: Factoring [10 points]. For this problem, let p and q be distinct large primes. Remember that finding roots of polynomials over the reals is easy.

(a) The best algorithm for factoring an RSA modulus $N = pq$ is the general number field sieve (GNFS). This algorithm runs in time

$$e^{\Omega((\log N)^{1/3}\cdot(\log\log N)^{2/3})}.$$

Compute the smallest integral constant $c \in \mathbb{Z}^+$ such that an RSA modulus of λ^c bits takes time $\Omega(2^\lambda)$ to factor.

[2] The big-$\tilde{O}$ and big-$\tilde{\Omega}$ notations are just like the normal big-O and big-Ω, except that they also suppress log factors. So, a running time of $\sqrt{q}\cdot\log^4 q$ is $\tilde{O}(\sqrt{q})$.

Note: The keys for standard elliptic-curve cryptosystems only need to be 2λ bits long to guarantee $\Omega(2^\lambda)$ security against the best known attacks. The answer to this question explains why elliptic-curve cryptosystems are much more popular than RSA today.

(b) Say that you are given an algorithm $\mathcal{A}$ that takes as input $N = pq$ and outputs $p + q$. Show that you can use $\mathcal{A}$ to factor N.

(c) Let f and g be fixed/public polynomials of constant degree with integral coefficients. In an attempt to shorten the size of your RSA secret key, you compute an RSA modulus using the following algorithm:

- Choose a random poly(λ)-bit integer x.
- If $f(x)$ and $g(x)$ are both primes, return $N = f(x) \cdot g(x)$.
- Else, restart.

Give an efficient algorithm that factors such a modulus N in polynomial time.