6.893: Cryptographic Techniques for Protecting Privacy **Fall 2020**

Problem Set 5

Due: November 13, 2020 at 5pm, Boston time via Gradescope.

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

`https://6893.csail.mit.edu/homework.tex`

Submission Instructions: You must submit your problem set via Gradescope. Please use course code **MGWNYV** to sign up. For ease of grading, please begin each *part* of each question on a new page.

Bugs: I 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: Extending Oblivious Transfers [20 points]. As we saw on the last problem set, oblivious transfer (OT) is an important building block of many secure MPC protocols. Because implementing OT requires public-key primitives, implementing a large number of OTs can be very expensive in practice. In this problem, you will see how to realize $n = \mathrm{poly}(\lambda)$ instances of 1-out-of-2 OTs on ℓ-bit strings (where $\ell = \mathrm{poly}(\lambda)$) using just λ instances of 1-out-of-2 OTs on λ-bit strings. Here, $\lambda \in \mathbb{N}$ is a security parameter. This means that we can essentially obtain an *arbitrary* polynomial number of OTs using a fixed number of *base OTs.*

(a) First, we show how to realize n instances of 1-out-of-2 OTs on ℓ-bit strings using λ instances of 1-out-of-2 OTs on n-bit strings (we refer to these as the *base OTs*). Consider the following protocol:

- Let $r \in \{0,1\}^n$ be the receiver's choice bits for the n OT instances, let $\big(m_0^{(1)}, m_1^{(1)}\big), \ldots, \big(m_0^{(n)}, m_1^{(n)}\big)$ be the sender's messages for the n OT instances.
- The receiver begins by choosing a matrix $\mathbf{M} \xleftarrow{R} \{0,1\}^{n\times\lambda}$. The sender chooses a random string $s \xleftarrow{R} \{0,1\}^{\lambda}$.
- The sender and the receiver now perform λ instances of an 1-out-of-2 OT on n-bit strings, but with their roles swapped (namely, the sender plays the role of the receiver in the base OTs, and vice versa). In the i^{th} base OT (where $i \in [\lambda]$), the sender provides s_i as its choice bit and the receiver provides $(\mathbf{M}_i, \mathbf{M}_i \oplus r)$ as its two messages, where $\mathbf{M}_i \in \{0,1\}^n$ denotes the i^{th} column of $\mathbf{M}$.
- Let $\mathbf{T} \in \{0,1\}^{n\times\lambda}$ be the matrix the sender receives from these base OTs (where the i^{th} column of $\mathbf{T}$ is the column it received from the i^{th} base OT). By construction, either $\mathbf{T}_i = \mathbf{M}_i$ or $\mathbf{T}_i = \mathbf{M}_i \oplus r$.

For $j \in [n]$, let $\mathbf{M}^{(j)}$ denote the j^{th} row of $\mathbf{M}$ and let $\mathbf{T}^{(j)}$ denote the j^{th} row of $\mathbf{T}$. Write down an expression for $\mathbf{M}^{(j)}$ in terms of $\mathbf{T}^{(j)}$ and s when $r_j = 0$ and when $r_j = 1$.

(b) Let $H\colon [n] \times \{0,1\}^{\lambda} \to \{0,1\}^{\ell}$ be a hash function (modeled as a random oracle). Suppose the sender encrypts each pair of messages $\big(m_0^{(j)}, m_1^{(j)}\big)$ by computing

$$\mathrm{ct}_0^{(j)} \leftarrow m_0^{(j)} \oplus H\big(j, k_0^{(j)}\big) \quad \text{and} \quad \mathrm{ct}_1^{(j)} \leftarrow m_1^{(j)} \oplus H\big(j, k_1^{(j)}\big),$$

where $k_0^{(j)}, k_1^{(j)} \in \{0,1\}^{\lambda}$. The sender sends each pair of encrypted messages to the receiver. Write down expressions for $k_0^{(j)}, k_1^{(j)}$ that would enable the receiver to learn $m_{r_j}^{(j)}$. [**Hint:** Recall that the receiver chooses the matrix $\mathbf{M}$ and then use the result from Part (a).]

(c) Give a brief explanation of why the receiver learns nothing about the other message $m^{(j)}_{1-r_j}$. [**Hint:** Use the fact that H is modeled as a random oracle.]

(d) Give a brief explanation of what goes wrong if we implement the hash function $H(j,k)$ as $F(k,j)$, where $F\colon \{0,1\}^\lambda \times [n] \to \{0,1\}^\ell$ is a secure PRF.

(e) Describe how you would modify the above protocol so that the base OTs are performed over λ-bit strings rather than $n = \text{poly}(\lambda)$-bit strings. [**Hint:** You will need to introduce a computational assumption.]

Problem 2: Constructing Linear PCPs [20 points]. Let $\mathbb{F}$ be a finite field. (As usual, think of $\mathbb{F}$ as the integers modulo some prime p.) Let $C\colon \mathbb{F}^n \to \mathbb{F}$ be an arithmetic circuit. We say that a pair of algorithms (P, V) is a fully linear probabilistically checkable proof (PCP) for C if it satisfies the following two properties:

- **Completeness.** For all $x \in \mathbb{F}^n$ such that $C(x) = 0$, if $P(x) \to \pi \in \mathbb{F}^m$, then $\Pr[V^{\langle \cdot, x \| \pi\rangle}() = \text{“accept”}] = 1$.
- **Soundness.** For all $x \in \mathbb{F}^n$ such that $C(x) \neq 0$, for all proofs $\pi \in \mathbb{F}^m$, $\Pr[V^{\langle \cdot, x \| \pi\rangle}() = \text{“accept”}] \leq O(|C|)/|\mathbb{F}|$.

Recall that the notation $V^{\langle \cdot, x \| \pi\rangle}$ means that the verifier V may issue queries of the form $q \in \mathbb{F}^{n+m}$ to its oracle, to which it receives answers $a = \langle q, x \| \pi\rangle \in \mathbb{F}$. You may assume that $|\mathbb{F}|$ is much larger than the number of gates in the circuit C.

(a) Show that there is a fully linear PCP for C in which (i) the verifier V makes n queries and (ii) the proof length $m = 0$.

(b) Assume that the circuit C consists only of *affine* gates—that is, only addition gates and multiplication-by-constant gates. Show that there is a fully linear PCP for C in which (i) the verifier V makes a single query and (ii) the proof length $m = 0$.

(c) Assume that the circuit $C : \mathbb{F}^n \to \mathbb{F}$ contains n^{100} affine gates and $\sqrt{n}$ multiplication gates. Show that there is a fully linear PCP for C in which (i) the verifier V makes $O(\sqrt{n})$ queries and (ii) the proof length $m = 0$.

(d) Let $C_{\mathsf{mul}} : \mathbb{F}^{3g} \to \mathbb{F}$ be a circuit that takes as input $3g$ values $(a_1, \cdots, a_g, b_1, \cdots b_g, c_1, \cdots, c_g) \in \mathbb{F}^{3g}$ and outputs 0 iff for all $i \in [g]$, $a_i \cdot b_i = c_i \in \mathbb{F}$. For this part, assume that there is a fully linear PCP for C_{mul} with proof size $O(g)$ and query complexity $O(1)$,

Let C be an arbitrary circuit with g multiplication gates. Show there is a fully linear PCP for C with proof size $O(g)$ and query complexity $O(1)$.

(e) Now that you have solved Part (d), you know that constructing a linear PCP for C_{mul} with proof size $O(|C_{\mathsf{mul}}|)$ and constant query complexity is sufficient for constructing a linear PCP for general circuits with proof size $O(|C|)$ and constant query complexity. This latter type of linear PCP is the one we used in class to construct SNARKs and zero-knowledge proofs on secret-shared data.

In the following, the notation $A(z) \in \mathbb{F}[z]$ indicates that $A(\cdot)$ is a polynomial in indeterminate z whose coefficients lie in $\mathbb{F}$.

We will now construct a linear PCP for C_{mul} in a few steps:

(i) Write the input to C_{mul} as $(a_1, \ldots, a_g, b_1, \ldots, b_g, c_1, \ldots, c_g) \in \mathbb{F}^{3g}$. Define $A(z) \in \mathbb{F}[z]$ to be the polynomial of lowest degree such that for all $i \in [g]$, $A(i) = a_i \in \mathbb{F}$. (As usual, we identify the elements of $\mathbb{F}$ with the integers modulo some prime p and we assume that $|\mathbb{F}| \gg g$.) Define a polynomial B similarly: for all $i \in [g]$, $B(i) = b_i$. Then, define a polynomial $C(z) = A(z) \cdot B(z)$. Explain why the following holds:

$$C_{\mathsf{mul}}(a_1, \cdots, a_g, b_1, \cdots b_g, c_1, \cdots, c_g) = 0 \quad \Leftrightarrow \quad \text{for all } i \in [g], \text{ it holds that } C(i) = c_i.$$

(ii) In the linear PCP for C_{mul} that we will construct, the linear PCP proof π_{mul} consists of the coefficients of the polynomials A, B, and C. How long is the proof, in terms of the number of field elements?

(iii) The verifier is given linear oracle access to: the input $(a_1, \ldots, a_g, b_1, \ldots b_g, c_1, \ldots, c_g) \in \mathbb{F}^{3g}$ and the coefficients of three polynomials A', B', C'. If the proof is honestly generated, then A', B', and C' are generated as in Part (i). If the proof is invalid (i.e., the prover is cheating), these polynomials might have some other form.

Prove that the verifier can use a *single* linear query to the input and proof to confirm (with very high probability) that $A'(i) = a_i$ holds for all $i \in [g]$. You should specify both the linear query that the verifier makes and the probability that an honest verifier fails to detect when there exists an $i^* \in [g]$ such that $A'(i^*) \neq a_{i^*}$.

(iv) Once you have the result of Part (iii), using two more queries, the verifier can also check that $B'(i) = b_i$ and $C'(i) = c_i$ for all $i \in [g]$. Use the Union Bound to upper bound the probability that an honest verifier fails to detect when any one of the three polynomials (A', B', C') disagrees with the inputs, in the sense of Part (iii).

(v) At this point, we have now verified the boundary conditions: that A', B', and C' encode the inputs to C_{mul}. Now we must check that these three polynomials satisfy $A' \cdot B' = C'$. Read about the Schwartz-Zippel Lemma. Explain how the verifier can test whether $A' \cdot B' = C'$ using three linear queries. Compute the probability that the verifier accepts even though $A' \cdot B' \neq C'$.

Discussion: Putting all of the parts together: The verifier knows that A, B, and C encode the a_is, b_is, and c_is. Furthermore, the verifier knows that $A \cdot B = C$, which implies that $c_i = a_i \cdot b_i$ for all $i \in [g]$.

At this point, you are done! Putting all of the parts together, you have constructed a fully linear PCP for general circuits with linear proof size and constant query complexity! Using the compilers we saw in class, you can now construct a SNARK or a zero-knowledge proof system on secret-shared data. This linear PCP construction is essentially the one that people use in practice, so you have reproved a non-trivial result.

(vi) Explain what goes wrong with the above construction if we take the field size $|\mathbb{F}| \approx g/2$.

(vii) **Extra credit [1 point]:** Show how to reduce the query complexity to 4.

(viii) **Extra credit [2 points]:** Explain why this fully linear PCP does *not* satisfy honest-verifier zero knowledge.

(ix) **Extra credit [3 points]:** Show how to reduce the proof size to $g - 1$.

Problem 3: Coppersmith Attacks on RSA [20 points]. This problem is not strictly about the material that we covered in class these past two weeks, but this class of attacks is so surprising that I think it's worth seeing here. This problem uses some basic facts about the RSA cryptosystem.

In this problem, we will explore what are known as "Coppersmith" attacks on RSA-style cryptosystems. As you will see, these attacks are very powerful and very general. We will use the following theorem:

> **Theorem** (Coppersmith, Howgrave-Graham, May)**.** Let N be an integer of unknown factorization. Let p be a divisor of N such that $p \geq N^{\beta}$ for some constant $0 < \beta \leq 1$. Let $f \in \mathbb{Z}_N[x]$ be a monic polynomial of degree δ. Then there is an efficient algorithm that outputs all integers x such that
>
> $$f(x) = 0 \bmod p \qquad \text{and} \qquad |x| \leq N^{\beta^2/\delta}.$$
>
> Here $|x| \leq B$ indicates that $x \in \{-B, \ldots, -1, 0, 1, \ldots, B\}$.

In the statement of the theorem, when we write $f \in \mathbb{Z}_N[x]$, we mean that f is a polynomial in an indeterminate x with coefficients in $\mathbb{Z}_N$. A *monic* polynomial is one whose leading coefficient is 1.

When $N = pq$ is an RSA modulus (where p and q are random primes of equal bit-length with $p > q$), the interesting instantiations of the theorem have either $\beta = 1/2$ (i.e., we are looking for solutions modulo a prime factor of N) or $\beta = 1$ (i.e., we are looking for small solutions modulo N).

For this problem, let N be an RSA modulus with $\gcd(\phi(N), 3) = 1$ and let $F_{\text{RSA}}(m) := m^3 \pmod N$ be the RSA one-way function.

(a) Let $n = \lceil \log_2 N \rceil$. Show that you can factor an RSA modulus $N = pq$ if you are given:

- the low-order $n/3$ bits of p,
- the high-order $n/3$ bits of p, or
- the high-end $n/6$ bits of p *and* the low-end $n/6$ bits of p.

(b) In the dark ages of cryptography, people would encrypt messages directly using F_{RSA}. That is, they would encrypt an arbitrary bitstring $m \in \{0,1\}^{\lfloor \log_2 N \rfloor / 5}$ by

- setting $M \leftarrow 2^{\ell} + m$ for some integer ℓ to make $N/2 \leq M < N$, and
- computing the ciphertext as $c \leftarrow F_{\text{RSA}}(M)$.

(Note that the first step corresponds to padding the message M by prepending it with a binary string "$10000 \cdots 000$.")

Show that this public-key encryption scheme is very broken. In particular, give an efficient algorithm that takes as input (N, c) and outputs m.

(c) To avoid the problem with the padding scheme above, your friend proposes instead encrypting the short message $m \in \{0,1\}^{\lfloor \log_2 N \rfloor / 5}$ by setting $M \leftarrow (m \| m \| m \| m \| m) \in \{0,1\}^{\lfloor \log_2 N \rfloor}$ and outputting $c \leftarrow F_{\text{RSA}}(M)$. Show that this "fix" is still broken.

(d) The RSA Full Domain Hash signature scheme uses a hash function $H : \{0,1\}^* \to \mathbb{Z}_N$. The signature on a message $m \in \{0,1\}^*$ is the value $\sigma \leftarrow F_{\text{RSA}}^{-1}(H(m)) \in \mathbb{Z}_N$. The signature σ is $n = \lceil \log_2 N \rceil$ bits long. Show that the signer need only output signatures of $2n/3$ bits while still

- retaining exactly the same level of security (i.e., using the same size modulus), and
- having the verifier run in polynomial time. We don't use this optimization in practice since (1) Schnorr signatures are so much shorter and (2) the verification time here is polynomial, but still much larger than the normal RSA-FDH verification time. Still, it's a cool trick to know.

Problem 4: Extra Credit: Zero-Knowledge Proofs on Distributed Data [3 points]. In class, we saw zero-knowledge proofs on secret-shared data, in which each of two verifiers holds additive shares (x_1, x_2) of an input $x \in \mathbb{F}^n$ (i.e., $x = x_1 + x_2 \in \mathbb{F}^n$) and the prover convinces them that $C(x) = 0$.

Sketch how to construct a zero-knowledge proof on *distributed data*, in which each verifier holds a piece of the input (i.e., $x_1, x_2 \in \mathbb{F}^{n/2}$) and the prover convinces the verifiers that $C(x_1 \| x_2) = 0$. Your proof system should require the prover to send $O(|C|)$ field elements to the verifiers, require the verifiers to exchange $O(1)$ field elements, and have soundness error $< 1/3$ (for a large enough field).

Problem 5: Extra Credit: Generic Discrete Log [6 points]. In this problem, we use the big-$\tilde{O}$ and big-$\tilde{\Omega}$ notations. These 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 $\tilde{O}(\sqrt{q})$.

Let $\mathbb{G} = \langle g \rangle$ be a group of prime order q. The best algorithm for discrete log that "works in all groups $\mathbb{G}$" (there is a way to make this statement precise) runs in time $\tilde{O}(\sqrt{q})$. This algorithm, which is shockingly simple, is the best known algorithm for discrete-log the standard elliptic curve groups we use in practice. The existence of a $\sqrt{q}$-time discrete-log algorithm is the reason why we use groups of order $q \approx 2^{256}$ in practice to achieve 128-bit security.

(a) 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 $\tilde{O}(\sqrt{B})$. (For an asymptotic treatment, we think of the bound $B = B(q)$ as some function of q.)

(c) [I do not know the answer to this problem.] Say that you are given the pair $(g, g^{x^2}) \in \mathbb{G}^2$ for $x \xleftarrow{R} \{0, \ldots, B\}$. Is it possible to recover x in time $\tilde{O}(\sqrt{B})$?

Problem 6: Feedback [1 points].

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

(b) What was your favorite problem on this problem set? In one sentence: why?

(c) What was your least favorite problem on this problem set? In one sentence: why?

(d) **[Optional]** If you have any other feedback on this problem set or on the course, please write it here or submit it using the anonymous feedback form.