CS 355: Topics in Cryptography **Spring 2024**

Problem Set

Due: Friday, 24 May 2024 (submit via Gradescope)

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

https://crypto.stanford.edu/cs355/24sp/homework.tex

Submission Instructions: You must submit your problem set via Gradescope. Please use course code **RKN4PX** 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.

Problem 1: Garbled Circuits are not maliciously secure [10 points]. Suppose that Alice has bit x, Bob has bit y, and they use Yao's GC protocol to compute $z = \mathrm{AND}(x, y)$.

(a) (True/False, no explanation): If Alice's bit is 1, and all parties follow the protocol, then afterwards Alice can tell whether Bob's bit is 0 or 1.

(b) (True/False, no explanation): If Alice's bit is 0, and all parties follow the protocol, then afterwards Alice can tell whether Bob's bit is 0 or 1.

(c) Suppose that Alice has bit 0 and plays the role of the garbler. Show that by incorrectly garbling the circuit, and by observing the response of Bob (who is following the protocol), Alice can learn Bob's bit. Explicitly describe how Alice's attack works, and informally explain why Bob cannot detect that Alice has deviated from the protocol. In your attack, Alice *must* participate honestly in the OT protocol.

Problem 2: Extending Oblivious Transfers [12 points]. Oblivious transfer (OT) is an important building block of many secure MPC protocols. Because OT requires public-key cryptography, implementing a large number of OTs can be very expensive in practice. In this problem, we will show how we can 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 $\left(m_0^{(1)}, m_1^{(1)}\right), \dots, \left(m_0^{(n)}, m_1^{(n)}\right)$ 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

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

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 $m_{1-r_j}^{(j)}$. [**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.

Problem 3: Secret Sharing and Polynomials [6 points.]. Let $\mathbb{F}$ be a finite field. (If it's helpful, you can think of $\mathbb{F}$ as the set of integers modulo a prime p.)

While working late into the night on your CS355 problem set, you miraculously discover a polynomial-time factoring algorithm! To safeguard your algorithm, you express it as a message $m \in \mathbb{F}$, and split it into n shares using the *additive* secret-sharing scheme we saw in lecture. You give one share s_i to each of your n friends (for $n \ll |\mathbb{F}|$) and then you spend many hours watching mind-numbing Netflix shows to erase the algorithm from your memory.

Later on, when you want to recover your secret message/algorithm m, you call your friends over to your dorm room. You ask each friend $i \in [n]$ to announce their share $s_i \in \mathbb{F}$ one at a time. Given these shares, you and your friends can all use the reconstruction algorithm to recover $m \in \mathbb{F}$.

(a) In the setting described above, each friend $i \in [n]$ announces their share s_i one at a time. Unfortunately, you are unlucky and the last friend to announce their share is not your friend, but is actually a frenemy. Show that by deviating from the specified protocol, the frenemy can broadcast a malformed share $s_n' \neq s_n$ that will cause you to recover a message $m' \neq m \in \mathbb{F}$, for any message m' of the frenemy's choosing.

(b) One (ineffective) way to prevent the attack would be to require all of your friends to announce their shares simultaneously. Show that the frenemy can still cause you to recover the incorrect message $m' = m + \Delta \in \mathbb{F}$, for any $\Delta \in \mathbb{F}$ of the frenemy's choosing.

Problem 4: Set Accumulators from Polynomial Commitments [8 pts]. Cryptographic accumulators represent a data structure as a small digest in such a way that operations over the data structure can be verified with access to only the digest. A *set accumulator* over universe $\mathcal{U}$ comprises five algorithms:

- $\mathsf{Create}(S \subset \mathcal{U}) \to d$: Creates a digest representing the set $S \subset \mathcal{U}$.

- $\mathsf{ProveMem}(S \subset \mathcal{U}, d, x \in S) \to \pi$: Creates a proof that $x \in S$.
- $\mathsf{VerifyMem}(d, x, \pi) \to \{0,1\}$: Verifies a proof that $x \in S$.
- $\mathsf{ProveNonMem}(S \subset \mathcal{U}, d, x \notin S) \to \bar{\pi}$: Creates a proof that $x \notin S$.
- $\mathsf{VerifyNonMem}(d, x, \bar{\pi}) \to \{0,1\}$: Verifies a proof that $x \notin S$.

A set accumulator is *membership-secure* if forging membership proofs for elements not in the set is hard. Similarly, a set accumulator is *non-membership-secure* if forging non-membership proofs for elements in the set is hard.[1] That is, if for all efficient adversaries $\mathcal{A}$, the following probabilities are negligible in λ:

$$\Pr\left[\begin{array}{l}(S,x,\pi) \leftarrow \mathcal{A}(\mathcal{U}) \\ d \leftarrow \mathsf{Create}(S)\end{array} : \begin{array}{c} x \notin S \wedge \\ \mathsf{VerifyMem}(d,x,\pi) = 1\end{array}\right] \quad \Pr\left[\begin{array}{l}(S,x,\bar{\pi}) \leftarrow \mathcal{A}(\mathcal{U}) \\ d \leftarrow \mathsf{Create}(S)\end{array} : \begin{array}{c} x \in S \wedge \\ \mathsf{VerifyNonMem}(d,x,\bar{\pi}) = 1\end{array}\right]$$

Using an evaluation-binding polynomial commitment scheme (which may, or may not be the KZG scheme) for polynomials over $\mathbb{F}$, build a set accumulator for $\mathcal{U} = \mathbb{F}$. Assuming the underlying polynomial commitment scheme has constant-size commitments and proofs, your digest, membership and non-membership proofs should be constant-size too. Prove that your accumulator is membership and non-membership secure, assuming that the underlying polynomial commitment scheme is evaluation binding.

Hint: What set of values is naturally associated with a polynomial?

Extra Credit [2 points]. Now, assume that your accumulator construction is instantiated with the KZG polynomial commitment scheme. Assume that there exists an efficient algorithm to compute the digest d' of $S \cup \{y\}$ given S, y and the digest d for S.

Show how one can compute a single update token, u, that allows any membership proof π for some x which is valid with respect to d to be updated into a new membership proof π' which is valid with respect to d'. That is, give two algorithms:

- $\mathsf{MakeToken}(S, d, y) \to u$: creates update token u for membership proofs with respect to d
- $\mathsf{UpdateProof}(d, x, \pi, u) \to \pi'$: creates a new proof π' which is valid with respect to d'

and show that these algorithms produce a valid π'.[2]

Problem 5: Understanding Zero Knowledge [15 points].

(a) Let $\mathcal{L}$ be an (arbitrary) NP language (with associated NP relation $\mathcal{R}$). Moreover, assume that all instances $x \in \mathcal{L}$ have a *unique* witness: namely, for every $x \in \mathcal{L}$, there is a unique w where $\mathcal{R}(x,w) = 1$. Give an interactive proof system for $\mathcal{L}$ that is complete and sound but is zero knowledge *if and only if* there exists a probabilistic polynomial time algorithm M that on input $x \in \mathcal{L}$ outputs the (unique) NP witness w for the instance x. $\big($Namely, $\Pr[w \leftarrow M(x) : \mathcal{R}(x,w) = 1] \geq \frac{2}{3}\big)$. Prove all the above properties.

[1] In the literature, both properties are called "collision resistance". We avoid that name here to avoid a confusion with hash functions.

[2] Note: Some accumulator constructions do not allow for the efficient update algorithms that this sub-problem requires. You may need to modify your accumulator so that it does allow for efficient updates.

(b) Suppose one-way functions exist. Give a language $\mathcal{L}$ and an interactive proof system for $\mathcal{L}$ that satisfies completeness, soundness, and honest-verifier zero knowledge, but *not* zero knowledge. Also prove all the above properties. [**Hint:** Recall from lecture that assuming one-way functions exist, the NP-complete language of graph Hamiltonian cycle has a zero-knowledge proof system. Modify this proof system so that the prover reveals additional information depending on the verifier's messages. To show that the resulting proof system is not zero-knowledge, show that the existence of a simulator (for a specific verifier) implies a decision algorithm for an NP-complete language. Use this to derive a contradiction.[3]]

Problem 6: Time Spent [1 point for answering]. How long did you spend on this problem set? This is for calibration purposes, and the response you provide will not affect your score.

Optional Feedback [0 points]. Please answer the following questions to help us design future problem sets. You do not need 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) What was your favorite problem on this problem set? Why?

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

(c) Do you have any other feedback for this problem set?

(d) Do you have any other feedback on the course so far?

[3] To formally show that a protocol is *not* zero-knowledge, you need to give a verifier V^* such that for *all* efficient simulators, the output distribution of the simulator is distinguishable from an interaction between the honest prover P and V^*.