Applied Cryptography and Security — March 8, 2024
Massachusetts Institute of Technology — 6.5610 Spring 2024
Recitation 5

Recitation 5: Fully Homomorphic Encryption and Complexity Review

1 Fully Homomorphic Encryption

This section is largely a repeat of the lecture notes.

Definition 1 *We define an **FHE scheme** with respect to a class of function circuits $\mathcal{C}$ to be the four randomized polynomial time algorithms* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *with security parameter $n \in \mathbb{N}$.*

- $\mathsf{Gen}(1^n) \to (\mathsf{sk}, \mathsf{ek})$
  - *outputs a secret decryption secret key* sk *and a public evaluation key* ek
- $\mathsf{Enc}(\mathsf{sk}, \mu) \to \mathsf{ct}$
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) \to \mu$
- $\mathsf{Eval}(\mathsf{ek}, F, \mathsf{ct}_1, \ldots, \mathsf{ct}_\ell) \to \tilde{\mathsf{ct}}$
  - *$F : \mathbb{Z}_2^\ell \to \mathbb{Z}_2$ is a member of our circuit class $\mathcal{C}$ that takes ℓ inputs, which are* $\mathsf{ct}_1, \ldots, \mathsf{ct}_\ell$.
  - $\tilde{\mathsf{ct}}$ *is an encryption of F evaluated on each (encrypted) input bit.*

In the above definition, note that only the user can encrypt and decrypt (they require sk)), but anyone can perform Eval. $\mathcal{C}$ is the class of circuits that we can homomorphically evaluate. In the Gentry-Sahai-Waters scheme we built in lecture, $\mathcal{C}$ is the class of Boolean circuits. We can represent any of these functions as layers of Boolean addition/multiplication gates. We wanted our FHE scheme to satisfy (1) Enc/Dec correctness, (2) semantic security, (3) compactness, (4) homomorphic addition, and (5) homomorphic multiplication. See the lecture notes for the formal definitions.

2 Complexity Review

Intuitively, a *language* or *decision problem* is a set of rules for which we either answer YES or NO on each input.

- e.g. Subset sum: Given a set S of positive integers and target value t, is there a subset $A \subseteq S$ such that $\sum_{a \in A} a = t$. Informally, the input space here is the pairs of (set of integers, integer).

- e.g. Halting problem: Given a program and its input, if we run the program on the input, will it halt?

We say a program A *decides* a language L if for all $x \in \mathcal{X}$ the input space, $A(x) = 1$ if $x \in L$ and $A(x) = 0$ if $x \notin L$.

2.1 Complexity classes

In lecture, the classes P, NP, IP, and $PSPACE$ were relevant to zero-knowledge proofs. A complexity class is a set of decision problems, and the relationships between classes describe the "hardness" of problems.

- **P**: The set of languages which are decidable in polynomial time. That is, all languages L such that there exists a program which decides whether $x \in L$ in poly(n) time, where $x \in \{0,1\}^n$.
- **NP**: The set of languages which are verifiable in polynomial time.
  - There exists a polynomial time algorithm called a verifier V, such that if $x \in L$, $V(x, w) = 1$ for *some* w and if $x \notin L$, $V(x, w) = 0$ for *all* w.
  - w is called a *witness* and can be thought of as a "proof" that x is in L.
  - e.g., in the subset sum problem, w would be a subset A such that its elements sum to t
- **IP**: A language $L \in$ IP if there is some pair of algorithms $(P, V(r))$ with the following property that decide L. For all candidates $x \in \mathcal{X}$,
  - V is poly-time having randomness r and P may be all-powerful.
  - P and V may exchange messages $m_1, \ldots, m_k$ for which each m_i may depend on the previous interaction $V(x, m_1, \ldots, m_k)$.
  - (Completeness) For all $x \in L$,

    $$\Pr[(P, V(r))(x) = 1] \geq 2/3$$

  - (Soundness) For all $x \notin L$ and cheating provers P^*,

    $$\Pr[(P^*, V(r))(x) = 1] \leq 1/3$$

    Here, a "cheating prover" just means a malicious prover who will try to convince the verifier that $x \in L$. P^* may be all-powerful as well.
- **PSPACE**: The set of languages which are decidable in polynomial space. That is, all languages L such that there exists a program which decides whether $x \in L$ in poly(n) memory, where $x \in \{0,1\}^n$. Note that programs for deciding languages in PSPACE can use exponential time!

Relationships between these classes will be written on the board.

Theorem 1 (Shamir '90) *IP = PSPACE*

This theorem is super cool because it tells us that interactive proofs may be more powerful than classical proofs. We know NP $\subseteq$ PSPACE, and if NP $\subset$ PSPACE=IP, then interactive proofs can decide a much larger class of languages than classical written proofs (which can prove NP).

2.2 Reductions and NP-Completeness

We use reductions to prove the relative hardness of problems in NP.

- Intuitively, we say that a language A reduces to a language B (denoted $A \leq B$) if we can utilize a program which decides B to decide A.
- Formally, a polynomial-time Karp reduction R is a polynomial time algorithm which takes a candidate x_A for A and produces $R(x_A) = x_B$ which is a candidate for language B. $x_A \in A \implies R(x_A) \in B$ and $x_A \notin A \implies R(x_A) \notin B$

We say a problem L is *NP-hard* if for all $A \in$ NP, $A \leq L$. We can think of L as being "as hard as" any problem in NP. We say L is *NP-complete* if $L \in$ NP and L is NP-hard. NP-complete problems can be thought of as the hardest problems in NP.

3 ZKP

Definition 2 (Zero knowledge proof) *An interactive protocol (P, V) is a* ***zero-knowledge proof system*** *for a language L if it is*

1. *Complete (as defined for IP)*
2. *Sound (as defined for IP)*
3. *Zero-knowledge. For every polynomial time V^* (which may be malicious), there exists an expected polynomial time simulator* Sim *such that for every $x \in L$:* Sim *and* $\mathsf{View}_{V^*}(P, V^*)$ *are computationally indistinguishable.*

That is, the verifier's view, or what it "knows," during the interaction can be simulated by itself in probabilistic poly time. This is how we define the notion of the verifier "learning nothing" from the interaction.

Theorem 2 (Goldreich-Micali-Wigderson '87) *Assuming one-way functions exist, all languages in NP have computational ZKPs.*

In lecture, we constructed a zero-knowledge proof for 3COL, which turns out to be an NP-complete problem. Therefore, there is a reduction from any $A \in$ NP to 3COL. So, informally, for any $A \in \mathrm{NP}$, we can construct a ZKP for A by computing the polynomial-time reduction from A to 3COL and performing the ZKP for the equivalent 3COL instance.

References to be added shortly.