COS 433 — Cryptography — Homework 11 — last one! :).

Boaz Barak

Total of 180 points. Due April 30th, 2010.

Since it seems several people had issues with zero knowledge (which is indeed a tricky concept), I decided to add another question on zero knowledge. Also, lectures 24 till 28 in Trevisan's lecture notes (`http://www.cs.berkeley.edu/~luca/cs276/`) are a good source for zero knowledge.

Exercise 1 (30 points)**.** A somewhat counterintuitive aspect of zero knowledge is that the simulator's job is just to sample a random variable and so it can generate randomness for the verifier in the process, which may seem "unfair". The following question illustrates why this is the right notion, by showing that the zero knowledge property implies that anything the verifier can learn about the secret after the interaction, he could have learned on his own. (This can be viewed as the mathematical formalization of Chazelle's description of zero knowledge..)

Recall the zero knowledge protocol for quadratic residuosity we saw in class (Protocol QR in notes for lecture 17).

1. Suppose that there is a cheating verifier V^* that satisfies the following property: if we choose $N = PQ$ as random Blum prime of $2n$ bits, choose $W \leftarrow_{\mathrm{R}} \mathbb{Z}_N^*$, and $X = W^2 \pmod{N}$, then after interacting with the prover for protocol QR on public input X, N and with prover's private input being W, V^* outputs the least significant bit of W with probability at least 0.99.

   Prove that in this case there is a polynomial-time algorithm A that if X, N, W chosen as above and A is given X, N as input, then A outputs the least significant bit of W with probability at least 0.98.

2. Prove that the above holds not just for that particular protocol, but for any ϵ-computational zero knowledge protocol (as per the definition given in class and also in homework 10).

Exercise 2 (30 points)**.** Prove that if there exists a fully homomorphic CPA-secure private key encryption scheme, then there exists a fully homomorphic CPA-secure public key encryption scheme. (The definition we use is with the $NAND$ algorithm, as in the lecture notes and homework 10.)

Exercise 3 (30 points)**.** In this exercise you'll prove the variant of the "leftover hash lemma" that we needed for showing that rerandomization works. This lemma is useful in many other settings, and so it's worthwhile to know in general.

1. Let $R \in \mathbb{N}$ and let $\pi : \mathbb{Z}_R \to [0, 1]$ be a probability distribution over $\mathbb{Z}_R$, that is $\sum_i \pi(i) = 1$. Let $cp(\pi)$ denote the collision probability of π. That is, $cp(\pi) = \Pr_{a,b\sim\pi}[a = b] = \sum_i \pi(i)^2$ (can you see why?). Prove that if $cp(\pi) \leq 1/R + \epsilon$ then the statistical distance of π from the uniform distribution is at most $10\sqrt{\epsilon R}$. See footnote for hint.[1]

[1]**Hint:** Write this statistical distance as $\sum_i |\pi(i) - 1/R|$ and use the Cauchy-Schwarz inequality that $\sum_i |a_i||b_i| \leq \sqrt{\sum_i a_i^2 \sum_j b_j^2}$.

2. Let $\overline{b} = b_1, \dots, b_{10n}$ and $\overline{a} = a_1, \dots, a_{10n}$ be two $10n$-bit vectors that are not the same (there is an i such that $a_i \neq b_i$). Prove that if R is prime and $Q_1, \dots, Q_{10n}$ are chosen at random in $\mathbb{Z}_R$ then $\Pr[\overline{Q} \cdot \overline{a} = \overline{Q} \cdot \overline{b}] = 1/R$, where $\overline{Q}$ denotes the vector $Q_1, \dots, Q_{10n}$ and $\overline{Q} \cdot \overline{a} = \sum_{i=1}^{10n} a_i Q_i \pmod{R}$. See footnote for hint[2]

3. For a vector $\overline{Q}$ as above, let $\pi_{\overline{Q}}$ denote the distribution over $\mathbb{Z}_R$ obtained by choosing a random set $S \subseteq [10n]$ (by setting each i to be inside S with probability $1/2$ independently) and outputting $\sum_{i \in S} Q_i \pmod{R}$. Prove that the expectation of $cp(\pi_S) - 1/R$ over the choice of a random Q, is at most 2^{-5n}.

4. Show the lemma stated in class: if $\overline{Q}$ is chosen at random as above, then with probability at least $1 - 2^{-n}$, the statistical distance of $\pi_{\overline{Q}}$ and the uniform distribution over $\mathbb{Z}_R$ is at most 2^{-n}. This completes the proof of the rerandomization step we showed in class, since now the distribution $\sum_{i \in S} Q_i P \pmod{N}$ for a random subset S will be statistically close to the distribution QP where Q is chosen uniformly over $\mathbb{Z}_R$.

Exercise 4 (30 points)**.** In this exercise we use a slightly different definition of $\mathcal{E}^{\mathbf{E}}(b)$. We define $\mathcal{E}^{\mathbf{E}}(b) = \{X \in \mathbb{Z}_N : X = PQ + 2E + b \pmod{N} \text{ and } |2E + b| \leq \mathbf{E}\}$. Note that this is the same as we defined in class up to a factor of 3 in the noise, and so none of the discussion in class is affected by this. Define $\mathsf{Add}(Y_1, \dots, Y_k) = \sum_i Y_i \pmod{N}$ and $\mathsf{Mult}(Y_1, \dots, Y_k) = \prod_{i=1}^k Y_i \pmod{N}$. Prove that if $Y_i \in \mathcal{E}^{\mathbf{E}_i}(c_i)$ then $\mathsf{Mult}(Y_1, ldots, Y_k) \in \mathcal{E}^{\mathbf{E}_1 \cdots \mathbf{E}_k}(c_1 c_2 \cdots c_k)$ and $\mathsf{Add}(Y_1, \dots, Y_k) \in \mathcal{E}^{\sum_i \mathbf{E}_i}(c_1 \oplus \cdots \oplus c_k)$.

Let C be a Boolean circuit, composed of multiplication and addition gates modulo 2, that computes a function mapping $\mathsf{GF}(2)^m$ to $\mathsf{GF}(2)$. Recall that we can think of C as a labeled directed acyclic graph, with m sources corresponding to the inputs and also the constants 0 and 1, one sink corresponding to the output, and the other vertices correspond to the gates, and are labeled with either $\oplus$ or $\times$. We define the *degree* of an input or constant vertex to be 1, and define the degree of other vertices inductively: let v be a vertex and suppose that we already defined the degrees $d_1, \dots, d_k$ of the vertices that have a edges to v. If v is labeled with $\times$ then we define the degree of v to be $d_1 + \cdots + d_k$, and if v is labeled with $\oplus$ then we define the degree of v to be $\max\{v_1, \dots, v_k\} + 1$. The degree of the circuit is the maximum degree of all its vertices.

Let C be a circuit of degree at most d and size at most S. Suppose that we are given m ciphertexts $X_1, \dots, X_m$ where $X_i \in \mathcal{E}^{\mathbf{E}}(b_i)$, and we run the circuit C on these ciphertexts, by changing each $\oplus$ gate to a call of Add and each $\times$ gates to a call of Mult. Prove that we get as output X in $\mathcal{E}^{\mathbf{E}'}(b)$ where $\mathbf{E}' \leq (\mathbf{E} + S)^d$ and $b = C(b_1, \dots, b_m)$. See footnote for hint[3]

Exercise 5 (30 points)**.** Let $f : \mathsf{GF}(2)^\ell \to \mathsf{GF}(2)$ be any function. Prove that f has a circuit of degree at most 2ℓ and size at most $2^{3\ell}$. See footnote for hint.[4]

Exercise 6 (30 points)**.** Let $SUM_i : \{0,1\}^m \to \{0,1\}$ be such that $SUM_i(a_1, \dots, a_m)$ denote the i^{th} bit of the sum $\sum_j a_j$. That is, one can write $\sum_j a_j = \sum_i 2^i SUM_i(a_1, \dots, a_m)$. Give a circuit of $\text{poly}(m, 2^i)$ size and degree at most $100(\log m + 2^i)$ to compute SUM_i. If you want, you can follow the approach below (or you can use any other design).

1. Define

$$S_k(a_1, \dots, a_m) = \sum_{\substack{S \subset [m] \\ |S| = k}} \prod_{j \in S} a_j \pmod{2}$$

[2]**Hint:** this is the same calculation we have done in the past when discussing pairwise independent hash functions.

[3]**Hint:** Sort the circuit topologically, and prove by induction.

[4]**Hint:** You can express f as the polynomial $\sum_{y \in \mathsf{GF}(2)^\ell} f(y) \prod_{i=1}^{\ell} (1 + x_i + y_i) \pmod{2}$.

Prove that $SUM_i(a_1, \ldots, a_m) = S_{2^i}(a_1, \ldots, a_m)$. See footnote for hint.[5]

2. Prove that $S_k(a_1, \ldots, a_m) = \sum_{j=0}^{k} S_j(a_1, \ldots, a_{m/2}) \cdot S_{k-j}(a_{m/2+1}, \ldots, a_m)$.

3. Give a circuit of $\text{poly}(m, 2^i)$ size and degree at most $100(\log m + 2^i)$ to compute the function SUM_i. (If it simplifies matters, you can assume m is a power of 2). See footnote for hint.[6]

By combining Exercises 4, 5 and 6, we can obtain the Clean procedure. The main goal there was to add up m numbers that are of $O(\log k)$ size, at most k of them are non zero. The idea is to follow the trivial gradeschool algorithm for addition. Recall that it starts by summing up the least significant digit of all the numbers, where the functions $SUM_1, \ldots, SUM_{logk}$ allow us to compute the carry. We then sum up the second least significant digit of all the numbers and add the first bit of carry from the first sum, and continue in this way. One can see that the i^{th} digit of the sum can always be computed with degree roughly 2^i.

[5]**Hint:** use induction on m

[6]**Hint:** use dynamic programming and the recursion above.