

## CAS CS 538. Problem Set 9

**Due 5pm Friday, December 10, 2010, by email to reyzin@bu.edu or in my mailbox in room MCS 138.**

**Problem 1.** (35 points) In this problem you will demonstrate how much easier life is in the random oracle model. Let  $\{f_i : D_i \rightarrow D_i\}$  be a trapdoor permutation family (which, of course, comes with the following probabilistic polynomial-time algorithms: the algorithm GenT to generate  $i$  and trapdoor  $t$ , an algorithm to compute  $f_i(x)$  given  $i$  and  $x \in D_i$ , and an algorithm to compute  $f_i^{-1}(y)$  given  $t$  and  $y$ ). Let  $H$  denote the random oracle. Let (Gen, Enc, Dec) be the following encryption scheme:

- Gen picks a trapdoor permutation: runs GenT to generate  $\text{PK} = i$  and  $\text{SK} = t$
- Enc( $\text{PK}, m$ ) picks a random  $r \in D_i$ , computes  $y = f_i(r)$ ,  $p = H(r)$ ,  $d = p \oplus m$ , and outputs  $c = (y, d)$ .
- Dec( $\text{SK}, c$ ) computes  $r = f_i^{-1}(y)$ ,  $p = H(r)$  and outputs  $m = d \oplus p$ .

Show that this encryption scheme is polynomially secure in the random oracle model, by reduction to the one-wayness of  $\{f_i\}$ . I will help you with the reduction by providing an intermediate step. If you can't prove the intermediate step in part (a), you can anyway use it in part (b).

**(a)** (20 points) Let  $D$  be the distinguisher. Show that  $D$  queries  $H$  on  $r = f^{-1}(y)$  with probability at least  $\epsilon$ , where the probability is taken over random choice of  $f$ , random choices of  $D$ , random choice of whether to encrypt  $m_0$  or  $m_1$ , and random choices made during the encryption process.

Advice: Recall (Notes 5, Definition 3) that  $D$  produces  $(m_0, m_1)$ , then  $m_b$  (for  $b = 0$  or  $b = 1$ ) gets encrypted to get  $(y, d)$ , and then  $D$  has to guess  $b$  given an encryption  $(y, d)$  of  $m_b$ . The way we phrased Definition 3, we considered  $D$ 's advantage  $\epsilon$  to be the difference between probability that  $D$  outputs 1 when  $b = 0$  and probability that  $D$  outputs 1 when  $b = 1$ . For this part, it is easier to use the following, equivalent formulation:  $b$  is chosen at random, and the advantage of  $D$  is the probability that  $D$  outputs  $b$  minus  $1/2$ . You don't need to prove the equivalence of the two formulations.

**(b)** (15 points) Now complete the reduction.

**Problem 2.** (30 points) In this problem, you will show that interactive proofs cannot get us above  $NP$  without the use of randomness and a small probability of error.

First, basic definitions for those who have not seen them before formally (we have been using them in class, of course). Recall that an algorithm (equivalently, Turing machine)  $A$  runs in polynomial time if there exists a polynomial  $p$  such that for every input  $x$ ,  $A(x)$  stops after at most  $p(x)$  steps. Recall that a *language* is a set of strings and that an algorithm  $A$  *decides*  $L$  if  $A(x) = 1$  for every  $x \in L$  and  $A(x) = 0$  for every  $x \notin L$ . A language  $L$  is decidable in polynomial time if there exists an algorithm  $A$  that runs in polynomial time and decides  $L$ .  $P$  denotes the set of all languages that are decidable in polynomial time.

A language  $L$  is decidable in nondeterministic polynomial time if there exists a polynomial  $p$  and language  $L'$  of pairs  $(x, w)$  such that:

- for every  $x \in L$ , there exists  $w$  of length at most  $p(|x|)$  such that  $(x, w) \in L'$ ;

- for every  $x \notin L$  and for all  $w$ , the pair  $(x, w) \notin L'$ ;
- $L' \in P$

The value  $w$  is called a *witness* for the fact that  $x \in L$ .

The set of all languages decidable in nondeterministic polynomial time is called *NP*. Note that for *NP*, we have noninteractive proofs:  $w$  is simply the proof. The language of all mathematical theorems whose proofs we can hope to verify noninteractively (e.g., by reading a paper) is in *NP*. We are hoping interactive proofs can help us verify even more theorems.

**(a)** (15 points) Prove that if the verifier in an interactive proof system for a language  $L$  is a deterministic polynomial-time algorithm, then  $L \in NP$ . Note that we are not requiring the prover to even be polynomial-time here. Hint: the verifier is completely predictable, so the prover doesn't need to interact at all in order to see what will happen.

**(b)** (15 points) Suppose an interactive proof system for a language  $L$  is such that the verifier can never be fooled—i.e., never accepts  $x \notin L$ . Suppose that the verifier is a polynomial-time (possibly randomized) algorithm. Prove that  $L \in NP$ . Hint: consider the view of the verifier.

**Problem 3.** (35 points) Consider the following protocol for proving knowledge of discrete logarithms. (We never formally defined proofs of knowledge, but you'll show specific properties below that essentially make up the definition.)

In this set up, we have a prime  $p$  and generator  $g$  of a subgroup of  $\mathbb{Z}_p^*$  of order  $q$ , where  $q$  is prime. We have a prover  $P$  (Paul) who has a secret key  $x \in \mathbb{Z}_q$  and a public key  $y = g^x \pmod p$ . The verifier  $V$  (Valerie) knows  $y, p, g$ , and  $q$ . The prover wants to convince the verifier that he knows  $x = \log_g y$ , without revealing to her what that value is.

Consider the following protocol.

- $P \rightarrow V$ :  $P$  selects a random  $r \in_R \mathbb{Z}_q$ , computes  $R = g^r \pmod p$ , and sends  $R$  to  $V$ .
- $V \rightarrow P$ :  $V$  selects a random  $c \in_R \mathbb{Z}_q$  and sends it to  $P$ .
- $P \rightarrow V$ :  $P$  computes  $a = r + cx \pmod q$  and sends  $a$  to  $V$ .
- $V$  check:  $V$  accepts if and only if  $g^a \equiv Ry^c \pmod p$ .

Note that this protocol is not a proof of membership in any interesting language—the statement “ $y$  has a discrete logarithm” is not an interesting statement, because every value in the subgroup generated by  $g$  has a discrete logarithm. Its value is in the fact that it is a proof of knowledge.

- *Completeness.* (10 points) Show that if both  $P$  and  $V$  follow instructions, then  $V$  will accept.
- *Proof of Knowledge.* (15 points) We want show that  $P$  knows  $x$ . Informally, this means that if  $P$  has a good chance of success for a random  $c$ , then  $P$  contains  $x$  somehow. A bit more precisely, if  $P$  sends a particular  $R$  for which he can produce answers for at least two different values  $c$ , then we can compute  $x$  from its answers. Show that if  $(R, c_1, a_1)$  and  $(R, c_2, a_2)$ , for  $c_1 \neq c_2$ , are both conversations accepted by  $V$ , then  $x$  can be computed from the values observed by  $V$ . (Note that here you cannot assume anything about how  $P$  actually computes these values, since you don't know how a malicious  $P$  would behave. All you can use is what the verifier observes.)

- *Honest-Verifier Zero-Knowledge.* (10 points) Show that  $V$  learns nothing. Namely, show how anyone who knows  $p, g, q$  and  $y$  (but not  $x$ ) can generate, in polynomial time, triples  $(R, c, a)$  that are distributed *identically* to the triples observed by  $V$  during a true interaction with  $P$ . (FYI: this is called honest-verifier zero-knowledge because it does not address dishonest behavior by  $V$ : if  $V$  chooses  $c$  in some funny way that depends on  $R$ , we can't show that  $V$  learns nothing, though neither can we show that  $V$  learns anything useful.)

FYI: this protocol can be transformed into a signature scheme in the random oracle model.  $P$  becomes the signer.  $P$  computes  $R$  the same way, computes  $c = H(R, m)$  where  $H$  is hash function (modeled as a random oracle), and computes  $a$  the same way. The output of signature is  $R, a$ . Verification is checking if  $g^a = Rm^c$ , for  $c = H(R, m)$ . Thus, an on-line  $V$  is replaced by a random oracle to allow the signature verification to be noninteractive. This signature scheme is known as the Schnorr signature scheme (proposed by Claus-Peter Schnorr in Eurocrypt 1989). The US Government signature standard DSA (and its variant ECDSA) is based on similar ideas, though it has no provability even in the random oracle model.

An open problem is to prove the security of this signature scheme for any reasonable  $H$  that is not a random oracle.