CS276: Cryptography Due Date: November 5, 2015

Problem Set 4

Instructor: Alessandro Chiesa *GSI: Manuel Sabin*

Problem 1

The DSA (Digital Signature Algorithm) was proposed by the National Institute of Standards and Technology (NIST) in August 1991 for use in their Digital Signature Standard (DSS). In this problem, we try to break the DSA under various attacks. For starters, the DSA works as follows:

- **Key-Generation:** $G(1^n)$. Pick primes p, q such that $p = 2q + 1$. Pick a generator h of the multiplicative group $\mathbb{Z}_p^*$, and set $g = h^2 (\text{mod } p)$. (Thus, g is a generator of a subgroup of order q of $\mathbb{Z}_p^*$.) Pick a random element $x \in \mathbb{Z}_p^*$. The public (verification) key is (p, g, g^x), and the secret (signing) key is x.
- **To sign a message** m: Choose k at random in $\mathbb{Z}_q^*$. Let $r = g^k (\text{mod } p)$. Solve the equation $rx - ks \equiv m (\text{mod } q)$ to find (the unique) solution s. Output (r, s). (**Note:** k is kept secret.)
- **To verify a signature** (r, s) **of a message** m: Check if $(g^x)^r r^{-s} \overset{?}{\equiv} g^m (\text{mod } p)$. If this equation checks, then output "signature verified" else output "fail".

(Before starting on the problems, convince yourself that this is indeed a correct signature scheme; that is, that valid signatures will pass verification.)

(a) Why does the k need to be secret in the signing algorithm? Find a way to recover the secret signing key, given a single signature (r, s) of a message m, along with the randomness k used to generate the signature.

(b) Usually (in practice), the k used for signing is not generated completely at random, but by using a pseudorandom generator. Again, in practice, people use a simple "pseudorandom" generator such as the linear congruential generator. A linear congruential generator **LCG**, on input a seed s_0 outputs a sequence $(s_1, s_2, \ldots, s_n)$ such that $s_i = as_{i-1} + b (\text{mod } q)$. (where a and b are assumed to be public parameters).

Show that if the k's are generated using an **LCG**, then given *two* signatures (of two messages), one can recover the secret signing key. ("Lesson": Beware of using unproven, patched-up pseudorandom generators).

(c) Finally, assume that the k's are generated by a cryptographically strong pseudorandom generator (e.g. start with a truly random seed, and apply G to get a pseudorandom value twice as long as the k values. Use the first half as k_1 and the second half as the next seed.). Prove that the DSA remains secure.

(In other words, show that if there is an adversary A that breaks the DSA that uses a PRG, then there is an adversary B that uses the original DSA – one where k's are generated randomly)

Problem 2

Consider the following interactive commitment scheme based on any pseudorandom generator $G : \{0,1\}^n \to \{0,1\}^{3n}$ (which, recall, exists based on any one-way function). In this description, Alice plays the role of the sender (the person committing), and Bob is the receiver.

Let $b \in \{0,1\}$ be the input of Alice.

Commitment Phase:

> Bob selects a random string $R \leftarrow \{0,1\}^{3n}$ and sends R to Alice.
>
> Alice samples a random seed $s \leftarrow \{0,1\}^n$ for the PRG. If $b = 0$, Alice sets $C = G(s)$; if $b = 1$, Alice sets $C = G(s) \oplus R$. Alice sends C to Bob.

Reconstruction Phase:

> To decommit, Alice sends her input b together with the seed s to Bob, who accepts if the previously received commitment message C is equal to $G(s) \oplus b \cdot R$.

(a) Prove that the scheme is *computationally hiding.* That is, at the conclusion of the commitment phase, no PPT malicious Bob can guess with non-negligible advantage (over the random coins of Alice) whether Alice committed to 0 or 1.

(b) Prove that the scheme is *statistically* binding. That is, for all adversarial Alice (not necessarily computationally bounded), it holds in the reconstruction phase that

$$\Pr_{Bob's\ coins}[\text{ Alice outputs } (s, s') \text{ s.t. Bob accepts both } (0, s) \text{ and } (1, s')\ ] < \epsilon(n),$$

for some negligible function ϵ.

Problem 3

Consider the language $L = \{(p, g) \ : \ p \in \mathsf{PRIMES} \wedge g \in \mathsf{GEN}(\mathbb{Z}_p^*)\}$. Without using any complexity assumption, prove that there exists a zero-knowledge proof system for the language L.

Problem 4

Given a language L, for each $k \in \mathbb{N}$ define the two languages L_k and $\overline{L}_k$ as follows:

$$L_k = \{x \ : \ x \in L \wedge |x| = k\} \ ,$$
$$\overline{L}_k = \{x \ : \ x \notin L \wedge |x| = k\} \ .$$

Suppose that L has a zero-knowledge proof system (P, V). Moreover, suppose that some (possibly cheating) verifier $\widetilde{V}$ can use his view to convince a third party, i.e., there exists a probabilistic polynomial-time algorithm D_L and a positive constant c_L such that for all sufficiently large k the following conditions hold:

1. For all $x \in L_k$,
$$\Pr\left[D_L(x, P\widetilde{V}[x]) = 1\right] > \frac{1}{2} + \frac{1}{k^{c_L}} \ .$$

2. For all $x \in \overline{L}_k$ and for all probabilistic polynomial-time algorithms B,
$$\Pr\left[D_L(x, B(x)) = 1\right] < \frac{1}{2} - \frac{1}{k^{c_L}} \ .$$

Prove that $L \in \mathsf{BPP}$.

Remark. This problem answers the question of whether there is a possibly cheating verifier $\widetilde{V}$ that, after communicating with the prover P, is able to use his own view for $x \in L$ to convince a third party (who was not present when P and $\widetilde{V}$ communicated) that $x \in L$. In particular, this says that, if $L \notin \mathsf{BPP}$, then, even if $x \notin L$, $\widetilde{V}$ may still be able to generate a fake view for $x \in L$ that looks good to any third party; because of this, $\widetilde{V}$ will not be able to use his real view for x to convince anybody else.