Tel Aviv University 16/11/2016

Introduction to Modern Cryptography (0368.3049) – Ex. 2

Benny Chor and Orit Moskovich

Submission in singles or pairs to Orr Fischer's Schreiber mailbox (289) until 30/11/2016, 23:59 (IST)

- Appeals/missing grade issues: bdikacs AT gmail.com
- Issues regarding missing/unchecked assignments will be addressed only if a soft copy will be submitted on time to: crypto.f16 AT gmail.com.
Subject of the email: Ex.2, ID

1. Recall the Discrete Log assumption (DL), Decisional DiffieHellman assumption (DDH) and Computational DiffieHellman assumption (CDH). In this question, we will prove the order of computational hardness of the three assumptions:

   Let G be a cyclic group of order n and $g \in G$ a generator.

   (a) If the DL problem is "easy" in G, then the CDH problem is "easy" in G:
   Let A be an efficient algorithm that given $h = g^x$, outputs x with probability ε (x was chosen uniformly from $\mathbb{Z}_{|G|}$). Show an efficient algorithm A' that given g^x, g^y, outputs g^{xy} with probability ε.

   (b) If the CDH problem is "easy" in G, then the DDH problem is "easy" in G:
   Let A be an efficient algorithm that given g^x, g^y, outputs g^{xy} with probability ε (x, y were chosen uniformly from $\mathbb{Z}_{|G|}$). Show an efficient algorithm A' that distinguishes between the two distributions

   $$D_0 = \{g^x, g^y, g^{xy} \mid (x, y) \leftarrow \mathbb{Z}_{|G|} \times \mathbb{Z}_{|G|}\}$$

   $$D_1 = \{g^x, g^y, g^z \mid (x, y, z) \leftarrow \mathbb{Z}_{|G|} \times \mathbb{Z}_{|G|} \times \mathbb{Z}_{|G|}\}$$

   with probability $\varepsilon - \frac{1}{|G|}$.

2. In this problem, we will show that any PRG $f : \{0,1\}^n \to \{0,1\}^{n+s}$ is a OWF.

   Assume A is an efficient algorithm that inverts f with probability $\geq \varepsilon$:

   $$Pr_{v \leftarrow f(U_n)}[A(v) \in f^{-1}(v)] \geq \varepsilon$$

   Show an efficient algorithm A' that distinguishes between $f(U_n)$ and U_{n+s} with probability $\geq \varepsilon - \frac{1}{2^s}$.

3. In recitation #1 we discussed the notion of Adversarial Indistinguishability, in the context of an unbounded adversary. In this question, we will consider computationally bounded adversaries.

   The eavesdropping indistinguishability experiment for adversary A:

   1) The adversary A outputs a pair of messages $m_0, m_1 \in \mathcal{M}$

   2) A random key k is generated using Gen, and a random bit $b \leftarrow \{0,1\}$

   3) The ciphertext $c = Enc_k(m_b)$ is computed and given to A

   4) A outputs a bit b'

   - We say that A wins $\iff b = b'$.

   Definition: An encryption scheme $\mathcal{E} = (Gen, Enc, Dec)$ has indistinguishable encryption in the presence of an eavesdropper if for all probabilistic polynomial-time (PPT) adversaries A

$$Pr[A \text{ wins}] \leq \frac{1}{2} + \varepsilon$$

   Construct the encryption scheme $\mathcal{E}$:

   Let $G : \{0,1\}^n \to \{0,1\}^\ell$ be a PRG. Define a private-key encryption scheme for messages of length ℓ as follows:

   - Gen: Choose $k \leftarrow \{0,1\}^n$ uniformly at random
   - Enc: On input a key $k \in \{0,1\}^n$ and a message $m \in \{0,1\}^\ell$, $Enc_k(m) = G(k) \oplus m$
   - Dec: On input a key $k \in \{0,1\}^n$ and a ciphertext $c \in \{0,1\}^\ell$, $Dec_k(c) = G(k) \oplus c$

   Prove: If G is a PRG, then $\mathcal{E}$ is a private-key encryption scheme that has indistinguishable encryption in the presence of an eavesdropper.

   Guidance:

   1) Assume there exists a PPT adversary A such that $Pr[A \text{ wins}] > \frac{1}{2} + \varepsilon$

   2) Construct a PPT adversary $D(w)$ that, using A, distinguishes between $G(U_n)$ and U_ℓ:

      - Run A to get two messages m_0, m_1

      - Generate an encryption of one of the messages and send to A

      - Obtain A's output and use it to identify the distribution

   3) Analyze the success probability of D: If $w \leftarrow U_\ell$, which encryption is generated?

      If $w \leftarrow G(U_n)$, which encryption is generated?

      In both cases, analyze the probability in which A wins the experiment.

4. Let $G : \{0,1\}^n \to \{0,1\}^{2n}$ be a PRG. Define the keyed function
$F : \{0,1\}^n \times \{0,1\}^{2n} \to \{0,1\}^{2n}$ as $F_k(x) = G(k) \oplus x$.
Prove that F is not a pseudorandom function (PRF).

   Guidance: Recall the definition for a PRF. Show a distinguisher between an oracle access to F and an oracle access to a random function, and use the oracle to query for more than one input.

5. (a) Is there a generic hardcore predicate for all one-way functions?
   (b) Let f be a function. Prove or dispute: f has a HCP implies that f is one-way.

6. Let p be a prime and a multiplicative generator $g \in \mathbb{Z}_p^*$.

   Define the function $f(x) = g^x \bmod p$.

   In this question we will prove that $parity(x) = x \bmod 2$ is not a HCP for f.

   Definition: An element $s \in \mathbb{Z}_p^*$ is a **Quadratic Residue** if there exists an element $r \in \mathbb{Z}_p^*$ such that $s = r^2 \bmod p$.

   Definition: $QR = \{s \in \mathbb{Z}_p^* \mid s \text{ is a quadratic residue}\} = \{s \in \mathbb{Z}_p^* \mid \exists r \in \mathbb{Z}_p^*.\ s = r^2 \bmod p\}$

   (a) Prove that QR is a sub-group of $\mathbb{Z}_p^*$
   (b) Let $g \in \mathbb{Z}_p^*$ be a generator. Prove that $g \notin QR$.
   (c) Let $g \in \mathbb{Z}_p^*$ be a generator. Prove: $\forall a \in \mathbb{Z}_p^*. a \in QR \iff a = g^{2k} \bmod p$ for some k
   (d) Prove Euler's criterion: Let $a \in \mathbb{Z}_p^*$. $a \in QR \iff a^{\frac{p-1}{2}} = 1 \bmod p$
   (e) Conclude that given $f(x) = g^x \bmod p$, it can be efficiently determined $parity(x) = x \bmod 2$.

7. In this question, you will implement two LFSRs and examine their cycle lengths. You should use either straight Python or Sage.

   (a) Consider the following LFSR1 with 17 bit internal state, where `feedback_bit = (state[0]+state[2]+state[3]+state[5]) % 2`. Let the initial state be `[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]` (written as Python list, so `state[0]=1` and for all other indices `state[i]=0`). Print the first 30 bits that are output by LFSR1 (with the first output bit being the leftmost bit). Figure out (namely ask your code to figure out) how many steps does it take till an internal state is repeated. Compare this to the maximum theoretical length, $2^{17} - 1$. Attach a printout of your code.
   (b) Do the same for LFSR2, with 17 bit internal state and the same initial state, where `feedback_bit = (state[0]+state[8]) % 2`.
   (c) Optional: Suggest an explanation for the difference between the two cycle lengths (this was not covered in class).