Cryptology I (spring 2014) Prof. Dr. Dominique Unruh

Exercise Sheet 2

Out: March 11, 2014 Due: March 24, 2014

You will need 50% of all homework points to qualify for the exam. (That is, if you get at least 50%, your final grade will be the exam grade. And if you do not get 50%, you do not pass the course.)

You may hand in your solutions in person or by email. If you submit by email, either scan a handwritten solution or typeset your solution readably. I do not consider ASCII formulas readable.

When submitting, indicate your name and your matriculation number. On your first submission, please also indicate a password, this password will be needed for accessing the solutions and your points online.

Problem 1: Reduction proofs

Let $G : \{0,1\}^n \to \{0,1\}^{2n}$ be a pseudo-random generator. Let $G_1 : \{0,1\}^{2n} \to \{0,1\}^{3n}$ be defined as follows: $G_1(x\|y) := G(x)\|y$ for $x, y \in \{0,1\}^n$. (Here $\|$ denotes concatenation.)

(a) Present a reduction showing that if G is a pseudo-random generator, then G_1 is a pseudo-random generator.

More precisely, given an adversary A that breaks G_1, construct an adversary that breaks G. (It is sufficient to draw a picture involving the challenger for pseudo-randomness, a "converter", and the adversary A, as we did in the lecture.)

(b) Download the file `prg.zip` from the lecture web page. It contains Java code that you can compile and run using "`javac Misc.java; java Misc`". Note that the source for class `Secret` and its subclasses is intentionally not provided and the class itself is obfuscated.

This code contains an implementation of a pseudo-random generator G (class `Secret.G`; you can invoke it using "`PRG g = new Secret.G(); g.random(seed)`") transforming 16 bytes of randomness into 32 bytes of pseudo-randomness.

Furthermore, it contains an implementation of the PRG G_1 from the previous problem (class `G1`) that uses G as a building block. (G_1 transforms 32 bytes into 48 bytes.)

Furthermore, it contains an adversary A_{G_1} (class `Secret.Adv_G1`) that breaks G_1. In other words, if we call "`PRGAdversary adv = new Secret.Adv_G1(); guess = adv.guess(random);`, then `guess` will be `true` iff `random` was produced by G_1 (with high probability).

Your task: Program an adversary `Solution.Adv_G_A` that breaks G by using the adversary A_{G_1}. Boilerplate code is already provided in `Solution.java`. Your adversary should guess right with probability > 0.95. Running `java Misc` will tell you how your adversary performed.

Note: Some utility functions for dealing with byte arrays are provided in `Misc.java`, see the source. In your solution, you are not allowed to cheat, e.g., by using reflection to access internal variables from the routine that tests your adversary. You are only allowed to use the adversary `Secret.Adv_G1`, but not `Secret.Adv_G2`.

(c) Explain why the fact that you found `Solution.Adv_G_A` "proves" the security of G_1 under the assumption that G is secure.

(d) **Bonus task (more tricky):** We have also implemented a second PRG G_2 (class `G2`) taking 16 bytes to 48 bytes. $G_2(s) := G(r_1)\|r_2$ where $r_1\|r_2 := G(s)$ ($G(s)$ is split into two 16 byte blocks r_1, r_2).

Two different adversaries against G_2 are provided, namely `new Adv_G2(true)` and `new Adv_G2(false)`.

Program an adversary `Solution.Adv_G1_B` using `Adv_G2(false)` and an adversary `Solution.Adv_G1_C` using `Adv_G2(true)`, both breaking G.

Each adversary should guess right with probability > 0.95. Running `java Misc` will tell you how your adversary performed.

Note: Some utility functions for dealing with byte arrays are provided in `Misc.java`, see the source. In your solution, you are not allowed to cheat, e.g., by using reflection to access internal variables from the routine that tests your adversary. You are only allowed to use the respective adversary `Secret.Adv_G1(true/false)`.

Problem 2: Linear block ciphers

Consider a block cipher E taking $2n$ bits to $2n$ bits that consists of a Feistel network with a round function $F(k, m^{half})$ that has the following property:

For each possible key k, there exists an $n \times n$ matrix $A^{(k)}$ such that for all n-bit messages m^{half}, we have $F(k, m^{half}) = A^{(k)} m^{half}$. (I.e., for a fixed key, F is linear.)

(Reminder: since we operate on bits, matrix multiplication is done modulo 2, i.e., $(A^{(k)} m^{half})_i = \sum_j A^{(k)}_{ij} m^{half}_j \bmod 2$.)

We will show that this block cipher cannot be secure under chosen plaintext attacks (by giving an attack).

Note: Even if you do not manage to solve the first parts of this problem, you can still solve the later parts by just assuming that you have solved the first ones.[1]

[1] The following argument is not allowed, though: "We show (c). Assume that I have solved (b). Then in my write-up, there would be a write-up of the solution of (b). Obviously, there is not. Thus we have a contradiction, false follows. From false anything follows, in particular (c). q.e.d."

(a) Let m be a $2n$-bit message within the Feistel network, before swapping the two halves. Construct a $2n \times 2n$ matrix B such that Bm is the result of swapping the two halves. (I.e., the first half of m is the second half of Bm and vice versa.)

(b) Let $m = m_1 \| m_2$ be a $2n$-bit message within the Feistel network. Let $m' = (m_1 \oplus F(k, m_2)) \| m_2$ be the result of applying F.

Construct a matrix $C^{(k)}$ such that $m' = C^{(k)} m$ for all m.

Hint: Try to first write down a formula (involving a sum) for the i-th bit of m'. Then you can read off the definition of $C^{(k)}$. $C^{(k)}$ may depend on $A^{(k)}$.

(c) Give a formula for the matrix $D^{(k)}$ such that $D^{(k)} m = E(k, m)$ for all m. $D^{(k)}$ may depend on B and $C^{(k)}$.

(d) Let m_i be the message $0 \ldots 010 \ldots 0$ with 1 on the i-th position. What is $D^{(k)} m_i$? How can we reconstruct the whole matrix $D^{(k)}$ given $D^{(k)} m_i$ for all i (efficiently)?

Note: You do not actually need the formulas derived in the previous steps. The only reason for finding those formulas was to show that the matrix $D^{(k)}$ exists at all.

(e) You are given a ciphertext $c = E(k, m)$ and you are allowed to make chosen plaintext queries (i.e., you may ask for $E(k, m')$ for any message m'). How do you find out m?

Hint: Find $D^{(k)}$ first.

(f) Explain why DES is susceptible to the attack described above if the S-boxes are linear.[2]

[2] I.e., for each S-box, each output bit is a linear combination of the input bits.