



KTH Computer Science
and Communication

Homework II, Foundations of Cryptography 2010

Before you start:

1. This homework set is due before March 26, 10:00. This is a strict deadline.
2. Read the detailed homework-rules at <http://www.csc.kth.se/DD2448/krypto10/rules>.
3. Read about B and H -points, and how these translate into grades, in the course description, http://www.csc.kth.se/DD2448/krypto10/course_description.pdf.

Preliminaries

Definition 1. The *Discrete Logarithm (DL)* assumption in a group G of prime order q states that if g and y are randomly chosen in G , then for every polynomial time algorithm A , $\Pr [A(g, y) = \log_g y]$ is negligible.

Definition 2. The *Diffie-Hellman (DH)* assumption in a group G of prime order q with generator g states that if $a, b \in \mathbb{Z}_q$ are randomly chosen, then for every polynomial time algorithm A , $\Pr [A(g^a, g^b) = g^{ab}]$ is negligible.

Definition 3. The *Decision Diffie-Hellman (DDH)* assumption in a group G of prime order q with generator g states that if $a, b, c \in \mathbb{Z}_q$ are randomly chosen, then for every polynomial time algorithm A , $|\Pr [A(g^a, g^b, g^{ab}) = 1] - \Pr [A(g^a, g^b, g^c) = 1]|$ is negligible.

Definition 4. The *RSA* assumption states that if $N = pq$, where p and q are randomly chosen primes with the same number of bits, $e \in \mathbb{Z}_{\phi(N)}^*$, and g is randomly chosen in \mathbb{Z}_N^* , then for every polynomial time algorithm A , $\Pr [A(N, e, g) = \beta \wedge \beta^e = g \pmod N]$ is negligible.

Definition 5. The *Strong RSA* assumption states that if $N = pq$, where p and q are randomly chosen primes with the same number of bits and g is randomly chosen in \mathbb{Z}_N^* , then for every polynomial time algorithm A , $\Pr [A(N, g) = (e, \beta) \wedge \beta^e = g \pmod N \wedge e > 1]$ is negligible.

Recall the notational conventions we discussed in class. The group G is for example formally a family $G = \{G_{q_n}\}$ of groups of prime orders q_n with $\lceil \log_2 q_n \rceil = n$, $g = \{g_n\}$ where g_n is a generator of G_{q_n} , and $q = \{q_n\}$. Thus, above we abuse notation and remove the index n from our notation, i.e., it is understood that G , q and g really means G_n , q_n , and g_n for increasing n .

Safe Primes

A prime p is said to be *safe* if $(p - 1)/2$ is prime as well.

Problems

- 1 The goal of this problem is to *prove* the following implications covered in class. In other words, the difficulty in solving this problem is not understanding that the implications hold, but to write down a *rigorous* proof. Thus, in this particular problem, any handwaving give zero points. A proof consists of a description of an efficient reduction and an analysis thereof.
 - 1a (3B) Prove that the Strong RSA assumption implies the RSA assumption.
 - 1b (3B) Prove that the DH assumption implies the DL assumption.
 - 1c (3B) Prove that the DDH assumption implies the DH assumption.
- 2 Consider the following physical key exchange protocol between Alice and Bob.
 1. Alice writes a random key $k \in \{0, 1\}^n$ on a piece of paper, places the paper in a coffin, and locks the coffin with her hanger lock, to which only she has the key. Then she sends the coffin to Bob.
 2. Bob locks the coffin with his own hanger lock to which only he has the key. Then he sends the coffin back to Alice. The coffin is now locked with both locks.
 3. Alice removes her own lock from the coffin and sends the coffin back to Bob. The coffin is now locked only using Bob's lock.
 4. Bob removes his lock from the coffin, and recovers the key k hidden in the coffin.

Consider next the following electronic analogon.

1. Alice chooses a joint key k and a "lock" r in $\{0, 1\}^n$ randomly and hands the "locked coffin" $c = a \oplus r$ to Bob.
 2. Bob chooses a "lock" s in $\{0, 1\}^n$ randomly and hands the "double-locked coffin" $c' = c \oplus s$ to Alice.
 3. Alice "removes her lock r ", $c'' = c' \oplus r$, and hands "coffin locked only with Bob's lock", c'' to Bob
 4. Bob "removes his lock s ", i.e., he computes $k = c'' \oplus s$.
- 2a (2B) Argue informally why the physical key exchange protocol is secure against a passive attacker that does not replace the coffin.
 - 2b (2B) Prove that the electronic analogon is insecure even against passive attackers that simply observe the exchanged messages.
 - 2c (1B) Discuss the difference is between the physical protocol and the electronic one that makes the latter insecure.

- 3 (10B) Let G_q be a cyclic group of prime order q and let g and h be randomly chosen generators of G_q . In class we defined Pedersen's hashfunction $h_{g,h} : \mathbb{Z}_q \times \mathbb{Z}_q \rightarrow G_q$ by $h_{g,h}(x, y) = g^x h^y$ and proved that it is collision-resistant under the DL assumption in G_q .

It is natural to generalize the construction to a function $h_{g_1, \dots, g_k} : \mathbb{Z}_q^k \rightarrow G_q$ defined by

$$h_{g_1, \dots, g_k}(x_1, \dots, x_k) = \prod_{i=1}^k g_i^{x_i}$$

where g_1, \dots, g_k are randomly chosen in G_q . Prove that for every constant k , the hashfunction h_{g_1, \dots, g_k} is collision resistant under the DL assumption in G_q . Hint: Given a pair (g, h) , your DL breaking reduction should replace two of the g_i with these generators, generate the rest of the g_i with known $\log_g g_i$, and then request a collision from the collision finder.

4

- 4a (5H) Use the prime number theorem to prove that if the RSA assumption holds, then it holds even if we pick the prime factors in the RSA modulus to be random *safe* primes of the same bit-length (instead of *any* primes of the same bit-length).

- 4b (5H) The proof carries over directly to the *strong* RSA assumption.

Prove that if the strong RSA assumption with safe primes holds, then this assumption holds even if in the problem instance (N, g) , the random generator g is chosen to have order $(p-1)(q-1)/4$.

- 5 Consider the hashfunction defined as follows. Let $N = pq$ where p and q are randomly chosen safe primes of the same bit-size, and let g be randomly chosen in \mathbb{Z}_N^* with order $(p-1)(q-1)/4$. Then define $h_{N,g}(x) = g^x \bmod N$.

- 5a (5B) Prove that a multiple of $(p-1)(q-1)/4$ can be computed from a collision.

- 5b (5B) Use this fact to prove that the hashfunction is collision-resistant under the strong RSA assumption. (You may use the conclusion of Problem 4 even if you did not solve it.)

- 6 (5H) Suppose that $h_\alpha : \{0, 1\}^{2n} \rightarrow \{0, 1\}^n$ is a collision resistant compression function when the index $\alpha \in \{0, 1\}^n$ is chosen randomly, and consider the hashfunction $h'_\alpha : \{0, 1\}^* \rightarrow \{0, 1\}^n$ defined as follows using a "hash tree". On input x with $t \times n$ bits, h'_α :

1. Finds the smallest integer l such that $t < 2^l$.
2. Forms $x_l = x \parallel 1 \parallel 0^{(2^l - t)n - 1}$, where \parallel denotes concatenation.
3. Note that x consists of 2^l blocks of n bits. If we hash the i th and $(i+1)$ th block of x for $i = 0, 2, 4, \dots, 2^l - 2$, we get 2^{l-1} blocks $y_0, \dots, y_{2^{l-1}-1}$, where $y_i = h_\alpha(x_{2i}, x_{2i+1})$. We repeat this hashing procedure iteratively until $l = 0$, in which case there is a single block which is taken to be the output of h'_α .

Is h'_α collision resistant? If so, then prove it. If not, then describe how to find collisions.

- 7 (10H) We consider elliptic curves $E_{a,b} : y^2 = x^3 + ax + b$ over \mathbb{Z}_{43} for various values of $a, b \in \mathbb{Z}_{43}$.
- 7a Which of the curves $E_{28,38}$, $E_{36,14}$, $E_{36,33}$, $E_{7,10}$, $E_{35,42}$, and $E_{10,24}$ are singular?
- 7b How many points are there on the curve $E_{7,37}$? (do not forget the point at infinity)
- 7c Figure 1 contains the affine part, i.e., everything except the special point at infinity, of the graph of $E_{6,21}$. Print a corresponding graph for the curve $E_{4,39}$.
- 8 In this problem we develop a pseudo-random generator which is provably secure under the DDH assumption. Let $p = 2q + 1$ be a safe prime with $\lfloor \log_2 q \rfloor = n$ and let G_q be the subgroup of quadratic residues in \mathbb{Z}_p^* .
- 8a (5H) Suppose that u is a random element in G_q . Prove that $u' = \min\{u, p-u\} \bmod q$ is a randomly distributed element in \mathbb{Z}_q , i.e., to generate a random element in \mathbb{Z}_q we can pick a random element u in G_q and then output u or $p - u$ modulo q depending on which is smallest.
- 8b (5H) Suppose that u' is randomly distributed in \mathbb{Z}_q and consider $u'' = u' \bmod 2^{n-t}$. Prove that $\sum_{s \in \{0,1\}^{n-t}} |\Pr[u'' = s] - 2^{-(n-t)}| < O(2^{-t})$, i.e., to generate an almost randomly distributed $(n-t)$ -bit string, we can pick a random $u' \in \mathbb{Z}_q$ and keep the $n - t$ least significant bits.
- 8c (5H) Suppose that you are given a random seed (g, x, r) , where $g \in G_q$ and $x, r \in \mathbb{Z}_q$. Describe a pseudo-random generator that stretches the random seed to a string of $10n$ bits. Hint: Iterate the map $f_{g,x}(r) = (g^r, g^{xr})$ and in each iteration output some of the output and keep the rest for the next iteration.
- 8d (10H) Prove the security of your pseudo-random generator under the Decision Diffie-Hellman assumption in G_q . (Hint: First use a hybrid argument to prove that you can stretch your seed to sequence of pseudo-random elements in G_q , then convert this sequence into a bit-string using the subproblems above.)

9 Implement your own El Gamal cryptosystem. Your code must be written in Java, C, or C++ and it must be packaged as follows and then emailed to `dogcsc.kth.se` with the subject `Krypto10:P9`.

1. Create a directory named `<lastname>_<firstname>`, where `<lastname>` and `<firstname>` are your last and first name respectively.
2. Put all your source files in the directory and add a README-file where the purpose of each source file is briefly explained.
3. Pack the directory using some standard archiving/compressing program, e.g., tar or gzip.

In this problem you may use an existing library for performing arithmetic with large integers, e.g., the Java class `java.math.BigInteger` or the C-library GMP, www.gmplib.org. However you may *not* use the existing implementations of modular exponentiation or primality tests (you may use them for debugging your own code though). Note that you are allowed to use existing routines for generating strong randomness, e.g., reading from `/dev/urandom` is allowed.

9a (8B) Implement modular exponentiation using the square-and-multiply algorithm.

9b (8B) Implement either the Miller-Rabin or the Solovay-Strassen primality test.

9c (5B) Implement the El Gamal cryptosystem in the subgroup $G_q \subset \mathbb{Z}_p$ of prime order q , where $p = kq + 1$ is a randomly chosen prime and $k < 10$. Your implementation should have three functions that can be called from an application in dire need of a cryptosystem:

1. A key generation algorithm that takes a single key length parameter n as input and outputs $((p, g, y), (p, g, x))$, where $p = kq + 1$ is a random prime with $k < 10$, g is a random generator of G_q , $x \in \mathbb{Z}_q$ is a random exponent, and $y = g^x \bmod p$.
2. An encryption algorithm that takes a public key (p, g, y) and an element $m \in G_q$ as input and outputs an El Gamal ciphertext $(u, v) = (g^r \bmod p, y^r m \bmod p)$, where $r \in \mathbb{Z}_q$ is randomly chosen.
3. A decryption algorithm that takes a secret key (p, g, x) and a ciphertext (u, v) , where u and v are elements in G_q , as input and outputs the message $m = vu^{-x} \bmod p$.

9d (5B) The implementation requires plaintexts in G_q . One way to embed an arbitrary message $m \in \{0, 1\}^{n-t}$ into G_q is to repeatedly try to pad the message with random bits until the result belongs to G_q . More precisely, repeatedly choose $s \in \{0, 1\}^t$ until $m' = m||s$ belongs to G_q , where $||$ denotes concatenation. Since at least a fraction $1/10$ of the elements in \mathbb{Z}_q belong to G_q and m' is “fairly random” a heuristic analysis shows that an embedding is not found within k attempts with probability roughly $\approx (1 - 1/10)^{-k}$.

To solve this problem you must implement the embedding, which means that you must also implement a method for checking if an element belongs to G_q .

