KTH Computer Science and Communication

Homework II, Foundations of Cryptography 2009

Due on March 13 at 15.15. The general rules on homework solutions available on the course homepage apply. In particular, discussions of ideas in groups of up to at most three people are allowed but implementation should be done individually. How points translate into grades is described in the course memo.

On the implementation problems that involve large numbers you are allowed to you use premade routines for addition, subtraction and multiplication of large integers as well as the operation to reduce one integer modulo a different integer. You may not use routines for more sophisticated operations such as modular exponentiation and modular inversions.

Note that this homework set has a possible total score of 123 points as opposed to the promised 100 points. As the thresholds for grades do not change the extra problem is there to give the student a choice of which problems to solve and not to create an extra burden.

1 (10p) Study the elliptic curves defined by

$$y^2 \equiv x^3 + 2x + b \bmod 7$$

by hand for $b = 3$ and $b = 5$. Calculate all points and a complete "addition table" as given by addition rule as defined in the book and in class. Are they both addition tables for a group? Can you explain what happened?

2 (15p) Consider the elliptic curve defined by

$$y^2 \equiv x^3 + 871219x + 1201730773687975 \bmod p$$

where $p = 1956804417664273$ is prime. It is of order $q = 1956804417666157$ (which is also prime) and generated by the point $G = (123456789100, 654321)$. Find a point P on the curve such that the first 11 digits of the x-coordinate is your 10-digit social security number ("personnummer") followed by a 0 (5p). Now find the discrete logarithm of this point relative to the generator G (10p). In other words find the number a, $0 \leq a < q$ such that $aG = P$.

3 (15p) A central concept of cryptography is that of a one-way function which is a function that is easy to compute (given x it is easy to find $f(x)$) and hard to invert (given y formed as $y = f(x)$ it is hard to find any x' such that $y = f(x')$[1]

If f maps n-bit strings to n-bit strings then we can invert f using 2^n evaluations of f and very little memory by simply evaluating f on all inputs. This inversion process can be sped up if we allow large amounts of memory and massive precomputation.

Your task is to find out how this works. Write a short summary of what is known both in the case when f is a permutation and the more general case when f may map different n-bit inputs to the same n-bit output (on thus some n-bit outputs are impossible). Estimate what size n you can attack with an ordinary computer in a week in the two situations. You should here ignore the cost of the precomputation. For the sake of concreteness let us assume that an ordinary computer has 2 GB of RAM, 500 GB of hard disc space and can compute the value of f on any input in 10^{-7} seconds.

Feel free to use any source of information on this problem except for the estimation problem where you should do your own calculations based on your own parameter choices.

4 (18p) Implement Schnorr's signature scheme. Choose the parameter sizes so that you expect them to withstand attacks for at least 20 years from attackers willing to spend a total of 10^7 USD to buy hardware. A sound motivation for your parameters is worth 5p. Choose all parameters needed (public and private for one user) and sign the message "Send more money" for an additional 13p.

Your code should be handed in as an attachment to an email to johanh@csc.kth.se with subject line "krypto09-sch". The attachment should consist of a single file named ''`firstname_lastname.tar.gz`'', such that the command ''`tar xvfz firstname_lastname.tar.gz`'' produces a single directory named ''`firstname_lastname`'' containing your files. If you write your solution in C/C++, the directory must contain a Makefile such that `make` builds your program. Regardless of the programming language used, your directory must contain a file `README` containing a brief description of how your program can be invoked, including a few concrete examples that execute in a reasonable amount of time. Your code should obviously be structured and documented. You may use pre-made subroutines as discussed in the heading of this problem set.

5 (20p) On the course web page you have five sets of sequences, ser1, ser2, ser3, ser4 and ser5. In each you find five pairs of sequences. In each pair one of the sequences have been produced by a bad pseudorandom generator giving some nonrandom[2] property while the other is output from a much better generator. In each set, the same bad generator was used for the five bad sequences. Identify a nonrandom property for each set and identify which sequence in each pair was produced by the bad generator.

[1] Note that if f is not one-to-one we cannot ask an inverter to find x.

[2] Of course this notion is imprecise. We mean a fairly natural property that appears with very low probability (lower than 10^{-6} for a truly random sequence.

6 (15p) You run into a web-page where there is an system for RSA public key encryption that uses values (in hex)

$$N = a0346acc9bf5007d9449bb74c827e72d08c8f1997613c102d4d21dc9ae3884610f00ff$$
$$6c5171401e668dde129a8fdb149c222122d96f233865abb6126cdd5815$$

and

$$e = e73d8e40da57cccc6a0ce5f9499388202813654293d89a64f7da2a0b24180da1897c6$$
$$74e75061d8fb14e120e5081edd3d46fff955003b5efd5770945bc1ee88b.$$

As you interact with the page you are puzzled by the fact that, even though you know that this web-server is run on an ordinary computer, it seems to decrypt much faster than your are able to encrypt. Factor N! The numbers are available on the course home page. Pre-made subroutines are allowed to the extent discussed in the homework headings.

7 (15p) In this problem we study a hash-function h that maps inputs of any length to $\{0,1\}^m$. We proved in class that if a hash-function behaves completely randomly then after computing about $2^{m/2}$ values we expect to find $m_1 \neq m_2$ with $h(m_1) = h(m_2)$. How many hash-values are needed to find

1. m_1, m_2 with $h(m_1) = \overline{h(m_2)}$ (bitwise complement).
2. m_1, m_2, m_3 with $h(m_1) = h(m_2) = h(m_3)$.
3. m_1, m_2, m_3 with $h(m_1) \oplus h(m_2) = h(m_3)$ (bitwise exclusive-or).

In each case we do not allow $m_i = m_j$ for $i \neq j$. If you cannot prove a bound rigorously you can resort to heuristic arguments.

8 (15p) Suppose n is a product of two primes, and x is a number that is relatively prime with n, i.e. $\gcd(x, n) = 1$. Consider the following interactive protocol for proving that x is a non-square modulo n, i.e. that the equation $y^2 \equiv x \bmod n$ is not solvable.

1. The verifier picks a random bit b and a random number r, $1 \leq r \leq n$ that is relatively prime with n. If $b = 0$ then the verifier sends $r^2 \bmod n$ to the prover and if $b = 1$ it sends $xr^2 \bmod n$.
2. The prover responds with a bit b'.
3. The verifier accepts iff $b' = b$.

Prove that this protocol is complete and sound. To prove that it is complete you should prove that if x is a non-square then an all powerful prover can convince a verifier of this fact. Prove that it is efficient in that if the prover knows the factorization of n then it can be implemented efficiently. To prove that it is sound you need to prove that a prover cannot fool an honest verifier too often. To be more precise prove that if x is a square modulo n then, for any strategy of the prover, the probability that a verifier following the protocol accepts is bounded by $1/2$.

Finally discuss whether this protocol has the zero-knowledge property. Argue that if the verifier follows the protocol then in fact it learns nothing new. Discuss what can be said if the verifier deviates from the protocol. Can you see any way to make it zero-knowledge (not needed for a full score on the problem)?