Cryptography assignment 2

Ishan Agarwal
Sr.No 10939

April 2016

(Note: for ans 1 not every detail has been written out but the proof stated conveys the basic idea. The details of the actual proof would in any case be very similar to question 3)

Answer 1

Consider two algorithms, A_1 is attempting to break the security of the multi-user protocol and A_2 is trying to solve the CDH problem. We shall show that if A_1 is successful with non- negligible probability then so is A_1.

Let the total number of sessions be n. Let A_2 pick a session at random and make the users use g^x and g^y which were given to A_2 by it's own challenger as part of the CDH game.Let us call this particular session the protected session. Note that if A_1 queries for the session key of this session or if it tries to corrupt these two members then A_2 fails.

Assuming however that A_1 does not choose to corrupt these two members and chooses to attempt breaking the security for this same session (the protected one), and is successful in the attempt, then A_1 is able to distinguish H(k) from some random string, where H is the hash function used which we shall model as a random function and k is the session key for that session i.e g^{xy}.

But we know that H is modeled as a random function. Hence if $H(g^{xy})$ can be distinguished from some other random string, then g^{xy} must have been queried to the Hash function oracle.

Now we know that if A_1 succeeds, then at some point it must have queried the oracle for g^{xy}. A_2 shall simply maintain a list of all the values queried for and one of these must be the correct g^{xy}. Furthermore as we are working under Gap DDH there is access to an oracle who, given a proposed solution to the DDH problem for some g^x and g^y will verify whether the proposed solution really is g^{xy}. Hence A_2 can use this oracle to check all the group elements, A_1 queried for their hash values. Whichever is correct can be submitted as a

solution to the verifier of A_2

It remains to be shown that A_2 can simulate the game for A_1. But as long as A_1 does not ask the session key of the protected session and does not try to corrupt this protected session's participants, A_2 can clearly always accomplish this.

Hence the probability of success of A_2 is the probability of success of A_1 multiplied by the chance that the session in question will be left untouched. (note this is an approximation we leave out of consideration some small terms that can easily shown to be negligible.

Thus the probability of success is of the order of $\frac{p}{n}$ where p is A_1's chance of success. This is non-negligible if p is non-negligible. Thus we have completed the reduction and proved the contra-positive of the problem statement.

Answer 2

We assume that the DDH problem is hard and assume that the PRG used is close to a random function. Since the definition of a PRG is that within polynomial time, there is only a negligible advantage possible if one tries to distinguish it from a truly random function, hence we shall, for the purposes of the analysis, assume the PRG to behave as a truly random function.

We shall show that if there is an adversary A who can win in the IND-CPA game with some non-negligible advantage a then we can break the DDH problem with some degraded, but still not negligible, advantage.

The scheme works in the following way:

- Some cyclic group G with it's generator is made known to one and all.
- To encrypt the message m we choose a random group element, x, and make it the senders secret key. G^x is this persons public key. We encrypt m as $(g^r, H((g^x)^r) \oplus m)$ where H is the function which we are modelling here as random.
- To decrypt we take (c_1, c_2) and find $H(c_1^x) \oplus c_2$.

Say that A is an adversary who can win the IND-CPA game with probability $\frac{1}{2} + \epsilon$. Let B be an algorithm attempting to solve the DDH problem. B receives (g^x, g^y, g^z) and has to determine if $g^z = g^{xy}$. B provides A with g^x as the public key. Then A can encrypt whatever messages it likes and finally gives B a pair of messages, m_0 and m_1. B chooses one of these at random, say m_i. Then it returns to A $(g^y, H(g^z) \oplus m_i)$. Now A may encrypt further messages if it so wishes and finally tries to guess which of m_0 and m_1 did B send it after encryption as above. If A guesses correctly, B replies to it's own verifier that

$g^z = g^{xy}$ else it claims it has not been given a solution to the DHP by it's verifier.

Let us now calculate the chance B has of winning. There are two equally likely cases:

- $g^z = g^{xy}$: in this case B wins if A guesses correctly. But the game for A has been modeled exactly so the chance of A winning is $\frac{1}{2} + \epsilon$
- $g^z \neq g^{xy}$: In this case g^z is some randomly picked group element and H being a random function, $H(g^z) \oplus m$ is exactly like using a one time pad, hence the probability of B's success is exactly $\frac{1}{2}$.

As both cases are equally likely, the final probability for B to win is $\frac{1}{2} + \frac{\epsilon}{2}$. Thus we have shown that if the scheme is not IND-CPA, then the DDH problem is solvable with non-negligible advantage. Thus we have shown the contra-positive of the required statement.

Answer 3

We shall state an encryption scheme and prove it's security under the following two key assumptions:

- The RSA problem is hard.
- We can model the hash function as a truly random function.

We show that if the scheme we outline is not IND-CPA secure then the RSA problem would not remain 'hard'. Thus we shall, as usual, use a reduction to prove the contra-positive of the statement.

We first state the scheme.

- (n,e) is a public key and (n,d) is the private key. These parameters are just as in the usual plain RSA scheme. There is also a hash function H, which we model as a random function, which is publicly known.
- Encryption proceeds by choosing a random group element, r and outputting as the cipher text: $(r^e, H(r) \oplus m)$
- Decryption happens by taking (c_1, c_2) and outputting $H(c_1^d) \oplus c2$

Let A be the adversary playing the IND-CPA game and B be an adversary attempting to solve the RSA problem. B receives (n,e,z) from his verifier and has to find some such x, such that $x^e = y (\text{mod n})$. Now B plays the IND-CPA game with A. If A asks for the hash of some element z, then B checks if z is an e^{th} root of y. If it is B is done else it continues playing. It responds to the hash query with a random value. If encryption queries are made, B picks a z at random, checks if z is a solution, and if it is not it returns the encryption of m

as $(z^e, H(z) \oplus m)$.

Ultimately A will give two messages, m_0 and m_1 to B. If no such z has been queried for such that $z^e = y(\text{mod n})$, then B can output $(y, R \oplus m_i)$, where R is some random number, as the e^{th} root of y could be anything at all. Then B continues answering queries from A. If no such z is queried for such that $z^e = y(\text{mod n})$, then B terminates unsuccessfully.

Consider now two cases (say $Z^e = y$):

- : A never queries for Z: Then the encryption for either m_0 or m_1 is essentially a one time pad and hence the probability of A winning the IND-CPA game is exactly $\frac{1}{2}$.
- A queries for Z: Then A will certainly recognize y as z^e and will surely win the IND-CPA game. Also B will win his game.

Let p be the probability of A winning the game and q be the probability of Z being queried. Then:

$$p = (1 - q).\frac{1}{2} + q.1$$

But we assumed that $p = \frac{1}{2} + \epsilon$ whereϵ is non-negligible (because we assumed that A will win IND-CPA game. Thus:

$$\frac{1}{2} + \epsilon = (1 - q).\frac{1}{2} + q.1 = \frac{1}{2} + \frac{q}{2}$$

Hence $q = 2\epsilon$ which is non-negligible as ϵ is non-negligible. Thus we have shown that, in a random hash function model, if he IND-CPA security is broken, the RSA problem too is not hard. Thus we have used the above reduction to prove the contra-positive of the given problem statement.

Answer 4

Consider FDH-RSA in which we model the hash function $H : \{0,1\}^n \to G$ as a random function. We shall further assume that the RSA problem is hard. With these two assumptions that FDH-RSA is EUF-CMA.

Our signature scheme is as follows:

- We use GenRSA algorithm as in the usual RSA to get a public key (N,e) and a private key (N,d).
- For any message m in $\{0,1\}^n$ the signer uses a Hash function H to compute h=H(m).
- The signature σ on the message m is then computed as $\sigma = h^d(\text{mod n})$.

- Verification is done by computing σ^e(mod n) and checking if this is equal to H(m).

Assume that there is some adversary A who can forge, onto some message m, a valid signature. We limit this adversary to making at most q_h queries to the Hash function oracle and q_s queries to the signature oracle. Without loss of generality we can consider the case where exactly q_h and q_s queries are made.

Consider another adversary, B, who has to break the RSA problem. He will be given (n,e,z) and has to find such a number x, such that $x^e = z$(mod n). We will show that if A can succeed at the forgery then B will succeed as well though with lower probability.

Now we begin the reduction. Let B answer the queries of A. We add the restriction that A cannot ask for the signature on any message m, without first querying the hash function oracle for H(m). We can ensure this because if A tries to do this B can simply make the hash function oracle query before accepting the signature query. Furthermore B stores the results of all these queries.

Now let us look at how B shall answer queries.

- For hash queries: At the i^{th} such query B randomly picks an x_i, such that $x_i \leq n$ and raises it to the e^{th} power modulo n. Let us call this number P_i. Now with some probability ρ, which we shall specify later, the algorithm B will output, as the Hash of m_i, this P_i. The rest of the time B will output $Q_i = z * P_i$(mod n). We are assuming H to be some random oracle and hence we can have B answer hash queries in this manner. B must also store each m_i and the P_i or Q_i which ever it had output.

- For signature queries: If the signature query is for some m_i for which the hash given was some x_i^e then we give x_i as the signature. This is a valid signature because $(x_i^e)^d = x_i$(mod n). If a signature query is made for a message where B answered in the form $Z * x_i^e$ B shall be unable to answer and shall abort. In this case B has clearly not succeeded.

After this process, A will return some message m and it's attempted forgery σ for that message. Now B will search it's list of messages queried and see if the message m was submitted for hash query. If it was and if B replied with $z * x_i^e$ then B is done. This is because a valid tag on such an m then $(z * x_i^e)^d = z^d * x_i$. If now B multiplies this signature with x_i^{-1} then he would obtain z^d which is an e^th root of z. However if this message had not been queried or if B had replied by simply raising some random x_i to the e^th power, then in this case too B will fail.

Let us now calculate the chance of B succeeding. B succeeds only if A never does a signature query on those messages to which B gave $z * x_i^e$ upon being hash queried as well as A finally provides a correctly forged signature for a message in B's list which B had claimed is hashed to some $z * x_i^e$. These two events are

independent.

The probability of B answering a hash query with something of the form x_i^e being ρ, then the probability of terminating successfully for B is clearly: $\rho^{q_s}.(1-\rho)$ To optimize this we set $\rho = \frac{q_s}{q_s+1}$ which gives us the maximum probability of success. This probability is:

$$(\frac{1}{q_s+1})(1-\frac{1}{q_s+1})^{q_s}$$

This is approximately $\frac{1}{e.q_s}$

Thus if A has an advantage of a then B has an advantage of approximately $\frac{a}{e.q_s}$