Information Security Lab - AS20

Reductions

Handout: Oct 16, 2020 12:00 AM

Due: Nov 2, 2020 3:30 PM

Exercise 3b -- Multi-User Ind-CPA Security of El Gamal PKE

Open Task

Exercise 3b - Tight Reduction from the Multi-User IND-CPA-security of ElGamal to DDH

In this exercise, we will look at the reduction that proves that El-Gamal is multi-user IND-CPA secure and that the reduction to DDH is tight.

Your task is to implement the `decideDDH` method of the `Solution3b` class:

```
public class Solution3b implements IDDHMultiUserIndCPAElgamalReduction {
    @Override
    public boolean decideDDH(DDHChallenge ddhChallenge, IElgamalMultiUserIndCPAAdversary adversary,
            IRerandomizationOracle rerandomizationOracle) {
            /*your code here*/
   return (true if DDH challenge is a DDH tuple, false if it is uniformly random);
  }
```

Your reduction should be tight, meaning you are allowed to query the `solveIndCPAChallenge` method of the `IElgamalMultiUserIndCPAAdversary` only **one time**.

Strategy of the reduction

The idea is to rerandomize the DDH challenge into multiple DDH challenges that will be used to construct ElGamal public keys and ciphertexts in the multi-user setting. In this exercise, you are **not** allowed to rerandomize the DDH challenge yourself, notice instead that the `decideDDH` method is given access to a rerandomization oracle which implements the interface `IRerandomizationOracle`. It is assumed that the oracle rerandomizes the DDH challenge correctly.

Supplementary information

An important thing to have in mind is that the adversary will be stateful. Let's look at the interface `IElgamalMultiUserIndCPAAdversary` implemented by our adversary against the multi-user IND-CPA security of ElGamal:

```
public interface IElgamalMultiUserIndCPAAdversary {
    int getNumberUsers();
    void init(IGroupElement generator, IGroupElement[] publicKeys);
    CandidateMessagePair<IGroupElement>[] getCandidateMessages();
    int solveIndCPAChallenge(ElgamalCiphertext[] ciphertexts);
}
```

`getNumberUsers()`

Method `getNumerUsers()` will return the number of users in the multi-user IND-CPA security game of this adversary.

`init(generator, publicKeys)`

`init` will start a multi-user IND-CPA game with this adversary. The adversary will save a copy of the given generator and public keys.

Calling `init` method a second time will cause this instance to overwrite its last saved values. If one of the inputs is not a valid group element, then this method will do nothing (the adversary will not save any values).

Important: The `IElgamalMultiUserIndCPAAdversary` adversary will only work correctly, if the public keys provided with `init` are exactly the second arguments of the DDH challenges you queried the last time from the rerandomization oracle.

This means, if you call `init(generator, publicKeys)` than `publicKeys` must be an array of the n elements $g^{d_1}, \ldots, g^{d_n}$ which come from the rerandomized DDH challenges $(g, g^{d_1}, g^{e_1}, g^{f_1}), \ldots, (g, g^{d_n}, g^{e_n}, g^{f_n})$ (in exactly this order) that you have received from your last `rerandomizationOracle.rerandomizeChallenge` call, where n is the number of users returned by `getNumberUsers()`.

If publicKeys is not equal to the second arguments of the rerandomized DDH challenges of your last rerandomizeChallenge call, then this method will do nothing. i.e., it will not save the generator and the public keys, and it will return null, when you call `getCandidateMessages`.

`getCandidateMessages()`

Call this method after you have called `init`. The adversary will generate n pairs of plaintexts (of type IGroupElement) and return them (where n is the number of users in the multi-user IND-CPA game of this adversary).

Now, let's take a closer look at the `IRerandomizationOracle` interface:

```
public interface IRerandomizationOracle {
    DDHChallenge[] rerandomizeChallenge(int numberOfRerandomizations, DDHChallenge originalChallenge)
            throws IllegalArgumentException;
}
```

The `rerandomizeChallenge` method rerandomizes a given DDH challenge. On input a number n and an original DDH challenge, this method returns n new DDH challenges. Those challenges have the same generator. Moreover, if the original challenge is of the form (g, g^x, g^y, g^z) and a returned DDH Challenge is of the form (g, g^d, g^e, g^f), then it holds that if $z = xy$, then $f = de$. Otherwise, if z is uniformly and independently distributed at random, then the elements g^d, g^e, g^f will also be uniformly and independently distributed at random.

This method guarantees that the second and third group elements of all returned DDH challenges are drawn uniformly and independently at random, saying that the exponents $d_1, \ldots, d_n, e_1, \ldots, e_n$ of all returned DDH challenges $(g, g^{d_i}, g^{e_i}, g^{f_i})$ are drawn uniformly and independently at random from $\{0, \ldots, |\mathbb{G}| - 1\}$.

Remember to use exactly the group elements $g^{d_1}, \ldots, g^{d_n}$ as public keys for the `IElgamalMultiUserIndCPAAdversary` adversary and do not rerandomize the DDH Challenge yourself.

Evaluation

When you submit your solution, our tests will run the reduction 200 times per adversary. Your reduction should win, on average, 75% of the games (meaning that the theoretical maximum advantage of your reduction should be 1/2). Your reduction passes our test if the advantage we measure is at least 25% (i.e. if it wins at least 62,5% of all games against each adversary).