Information Security Lab - AS20

Reductions

Handout: Oct 16, 2020 12:00 AM

Due: Nov 2, 2020 3:30 PM

Exercise 1 -- Onewayness of El Gamal PKE

Open Task

Hardness of CDH implies the OW-CPA Security of ElGamal

Task Description

Let $\mathbb{G}$ be a cyclic group (of prime order p), g a generator of $\mathbb{G}$, and recall the ElGamal Public-Key Encryption (PKE) scheme:

El-Gamal PKE

Setup($\mathbb{G}$)

$\mathsf{sk} \leftarrow_r \mathbb{Z}_p$
$\mathsf{pk} \leftarrow g^{\mathsf{sk}}$

Encrypt($m \in \mathbb{G}$)

$\mathsf{r} \leftarrow_r \mathbb{Z}_p$
$c_0 \leftarrow g^r$
$c_1 \leftarrow \mathsf{pk}^r * m$
Output (c_0, c_1)

Decrypt(sk, $(c_0, c_1) \in \mathbb{G}^2$)

$\mathsf{r} \leftarrow_r \mathbb{Z}_p$
Output $(c_1/(c_0)^{\mathsf{sk}})$

In this exercise, we show that if CDH is hard in the group $\mathbb{G}$, then ElGamal satisfies OW-CPA security. The OW-CPA game is defined as follows:

The OW-CPA Experiment for an encryption scheme (Setup,Enc,Dec)

$(\mathsf{pk}, \mathsf{sk}) \leftarrow_r \mathsf{Setup}(\mathbb{G})$
$m^* \leftarrow_r \mathcal{M}$ // $\mathcal{M}$ is the message space
$\mathsf{ct} \leftarrow_r \mathsf{Enc}(\mathsf{pk}, m^*)$
$m \leftarrow_r \mathcal{A}(\mathsf{pk}, \mathsf{ct})$
Output $(m^* = m)$

The reduction will receive and solve one CDH challenge, given **one** extraction query to an adversary that breaks the OW-CPA security of El-Gamal. Your task is to implement the `ICDHOwCPAElgamalReduction` method of the `Solution1` class:

```
public class Solution1 implements ICDHOwCPAElgamalReduction {
    @Override
    public IGroupElement solveCDH(CDHChallenge cdhChallenge, IElgamalOwCPAAdversary adversary) {
    //your code
    return (an IGroupElement object, your solution to the CDH challenge);
    }
}
```

Evaluation

When you submit your solution, our tests will run your reduction one hundred times. In this assignment, your reduction must be successful in all games, in order to be considered valid. The *run* button makes a quick test for your reduction (lasting several seconds), while the *test* button will perform more extensive tests, similar to the grading tests ran after pressing the *submit* button. These longer tests last up to 10 minutes.

Note: the grade that you receive in CodeExpert is preliminary, more extensive tests are performed after the deadline. You should think of potential special cases that potentially are not covered by a simple solution.

The evaluation environment limits the execution time of your programms to ten seconds of CPU time (which suffices for a correct solution). If your solution does not compile, the testing enviroment will not be able to grade it, which means it will be awarded 0 points by default.

Supplementary Information

You are not given access to the group representation, therefore your reduction should be generic with respect to the group operations. Now, we take a look at the `IGroupElement` interface:

```
public interface IGroupElement {
    BigInteger getGroupOrder();
    IGroupElement multiply(IGroupElement otherElement);
    IGroupElement power(BigInteger exponent);
    IGroupElement invert();
    IGroupElement clone();
}
```

This interface has five methods, which we describe presently. `getGroupOrder()` returns the order of the group, a `BigInteger` object. The `BigInteger` class is the standard java.math class, whose documentation can be found at: https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html.

Returning to IGroupElement, you may assume that objects of classes which implement this interface are immutable. For this reason, we need `clone`, which returns a copy of the object. Finally, `multiply` and `power` are used to multiply and exponentiate group elements.

Next, we take a look at the `ElgamalCiphertext` class. The constructor creates an empty ciphertext, where both `c0` and `c1` are initialized to `null`. When the ciphertext is well-formed, attributes `c0` and `c1` are of the form g^r and $\mathsf{pk}^r * m$ (notice that m is an `IGroupElement` object).

```
public class ElgamalCiphertext {
    public ElgamalCiphertext() {}
    public IGroupElement c0;
    public IGroupElement c1;
}
```

The public key **pk** is the public key of an instance of the class `ElgamalPKEScheme`, which we describe below:

```
public class ElgamalPKEScheme {
    private SecureRandom RNG;
    public KeyPair<IGroupElement, BigInteger> setup(IGroupElement generator) {
        KeyPair<IGroupElement, BigInteger> pair = new KeyPair<IGroupElement, BigInteger>();
        pair.secretKey = getRandomBigInteger(RNG, generator.getGroupOrder());
        pair.publicKey = generator.power(pair.secretKey);
        return pair;
    }
    public ElgamalCiphertext encrypt(IGroupElement generator, IGroupElement publicKey, IGroupElement me
        ElgamalCiphertext ciphertext = new ElgamalCiphertext();
        BigInteger r = getRandomBigInteger(RNG, generator.getGroupOrder());
        ciphertext.c0 = generator.power(r);
        ciphertext.c1 = publicKey.power(r).multiply(message);
        return ciphertext;
    }
    public IGroupElement decrypt(BigInteger secretKey, ElgamalCiphertext ciphertext) {
        return ciphertext.c1.multiply(ciphertext.c0.power(secretKey.negate()));
    }
}
```

RNG is an instance of `SecureRandom`, which is a cryptographically secure pseudo-random number generator. `Setup` takes as input an `IGroupElement` that is a generator of our group $\mathbb{G}$ and generates a key pair $(\mathsf{pk}, \mathsf{sk})$, represented by KeyPair<IGroupElement, BigInteger>. As explained before, `encrypt` takes as input a message m (m is an `IGroupElement` object) and computes $(g^r, \mathsf{pk}^r * m)$, where r is a uniformly random `BigInteger` from the group of exponents. Decryption simply computes $\mathsf{c}_1 * (\mathsf{c}_0^{-\mathsf{sk}})$.

In order to implement the reduction code in method `solveCDH(CDHChallenge cdhChallenge, IElgamalOwCPAAdversary adversary)`, we still need to explain how to use class `CDHChallenge` and interface `IElgamalOwCPAAdversary`. Class `CDHChallenge` has the following description:

```
public class CDHChallenge {
    public CDHChallenge() {}
    public IGroupElement generator;
    public IGroupElement groupElementX;
    public IGroupElement groupElementY;
}
```

You have access to the attribute `generator`, which represents the generator g of the CDH challenge. In the CDH game, `groupElementX, groupElementY` correspond to g^x, g^y, where $x, y \leftarrow_r \mathbb{Z}_p$ are uniformly random elements of $\mathbb{Z}_p$

Finally, let us look at the interface `IElgamalOwCPAAdversary`:

```
public interface IElgamalOwCPAAdversary {
    void init(IGroupElement generator, IGroupElement publicKey);
    IGroupElement extractMessage(ElgamalCiphertext ciphertext);
}
```

To start an OW-CPA game against ElGamal with the `IElgamalOwCPAAdversary` adversary given as a parameter to `solveCDH`, you need to call the `init` method. The adversary will save a copy of the given Generator and PublicKey of some instantiation of the ElGamal PKE. Calling this 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).

After `init`, we can ask the adversary to decrypt a ciphertext. For this, we can call the method `extractMessage`. This method will return a group element which is the message of the given ciphertext, if this adversary can successfully decrypt the given ciphertext.

You may need to generate your own uniformly random `BigInteger` or `IGroupElement` objects. As an example, we here explain one way you can generate your own uniformly random `IGroupElement` objects in $\mathbb{G}$:

```
SecureRandom RNG = new SecureRandom();
BigInteger r = getRandomBigInteger(RNG, generator.getGroupOrder());
IGroupElement c = generator.power(r);
```

IMPORTANT For this assignment, recall that you should only run the `extractMessage` method **once**.