Reduction of DLog to CDH (18 points)

In this task, your job is to show that you can solve the Discrete Logarithm problem (DLog) (for special prime orders) when given access to an oracle that solves the Computational Diffie Hellman (CDH) problem for you. For this end, you will finish the implementation of a generic reduction (`DLog_CDH_Reduction`) that can solve DLog-challenges when given access to a perfect CDH adversary. To simplify your task, the order of the group, in which you work, will always be a prime number with special properties, as we will explain later.

Below, we will explain various parts of this task and various objects, with which your implementation needs to work with.

Generic Group Model

In this task, you will work in the **Generic Group Model** (GGM). In the GGM, it is assumed that there is a cyclic group $\mathbb{G}$ of prime order p (where $p \in \Theta(2^\lambda)$) to which you only have *oracle access*. I.e., each element $g \in \mathbb{G}$ is given to you by a random binary string, a *handle*, which represents this element but does not leak any information about its exponent. Given two handles of group elements g, h in the GGM, you need to ask an oracle to receive a handle of the product $g \cdot h$ or of a power g^e for $e \in \mathbb{Z}$.

For this task, we have already implemented the GGM and will provide you with an interface that encapsulates all necessary functionalities:

```
public interface IGroupElement extends IBasicGroupElement<IGroupElement> {
    BigInteger getGroupOrder();
    IGroupElement multiply(IGroupElement otherElement);
    IGroupElement power(BigInteger exponent);
    IGroupElement invert();
    IGroupElement clone();
    boolean equals(IGroupElement otherElement);
}
```

Each group element that you are given in this task will be of the type `IGroupElement`. Use the methods of this interface to perform generic group operations.

Computational Diffie-Hellman Assumption

In this task, your job is to show that if the Discrete Logarithm problem (DLog) is hard in $\mathbb{G}$ then the Computational Diffie-Hellman problem (CDH) is hard, too. For this end, you will write a generic reduction that can solve DLog-challenges when given access to an adversary that can solve CDH-challenges.

Let us first define the CDH assumption (as in the lecture): Let $g \in \mathbb{G}$ be a fixed generator of the cyclic group $\mathbb{G}$ of order p. Draw $x,y\gets \mathbb{Z}_p$ uniformly at random. A CDH challenge is of the form

> (g, g^x, g^y).

The solution to the above CDH challenge is the group element

> g^{xy},

which is uniquely determined by the challenge (g, g^x, g^y). I.e., the CDH problem consists in multiplying group elements *in the exponent*.

Formally, the associated CDH challenge game looks at follows:

1. The CDH-challenger $\mathcal{C}$ fixes a generator g of the group $\mathbb{G}$ and draws $x,y\gets \mathbb{Z}_p$ uniformly at random.
2. $\mathcal{C}$ starts a DDH-adversary $\mathcal{A}$ and hands him over the CDH-challenge (g, g^x, g^y) when $\mathcal{A}$ asks for it.
3. $\mathcal{A}$ performs some generic computations on (g, g^x, g^y) and comes up with a group element g^z that it sends back to $\mathcal{C}$.
4. $\mathcal{C}$ checks if $g^z = g^{xy}$. If $g^z = g^{xy}$ then $\mathcal{A}$ wins. Otherwise, $\mathcal{A}$ loses.

The **advantage** of $\mathcal{A}$ in this security game is defined by
$$Adv_{CDH}(\mathcal{A}) := \Pr[\mathcal{A} \text{ wins}] =\Pr_{x,y\gets \mathbb{Z}_p}[\mathcal{A}(g,g^x,g^y) = g^{xy}] .$$
The **CDH-assumption** states that for each ppt adversary $\mathcal{A}$ its advantage in winning the above is negligible (in the security parameter).

To encapsulate CDH challenges in this task, we created a container class, which you should use:

```
public class CDH_Challenge<IGroupElement> {
    public final IGroupElement generator;
    public final IGroupElement x;
    public final IGroupElement y;
}
```

Discrete Logarithm Problem

The Discrete Logarithm Problem consists in finding the exponent of a group element relative to a generator of the group. I.e., given a generator $g \in \mathbb{G}$ and a group element h one should find an exponent $x \in \mathbb{Z}_p$ s.t.

> $g^x = h$.

The associated security game is given by:

1. The DLog-challenger $\mathcal{C}$ fixes a generator g of the group $\mathbb{G}$ and draws $x\gets \mathbb{Z}_p$ uniformly at random.
2. $\mathcal{C}$ starts a DLog-adversary $\mathcal{A}$ and hands him over the Dlog-challenge (g, g^x) when $\mathcal{A}$ asks for it.
3. $\mathcal{A}$ performs some generic computations on (g, g^x) and comes up with a number $y \in \mathbb{Z}_p$ that it sends back to $\mathcal{C}$.
4. $\mathcal{C}$ checks if $x=y$. If $x=y$ then $\mathcal{A}$ wins. Otherwise, $\mathcal{A}$ loses.

The **advantage** of $\mathcal{A}$ in this security game is defined by
$$Adv_{Dlog}(\mathcal{A}) := \Pr[\mathcal{A} \text{ wins}] =\Pr_{x\gets \mathbb{Z}_p}[\mathcal{A}(g,g^x) = x].$$
The **DLog-assumption** states that for each ppt adversary $\mathcal{A}$ its advantage in winning the above is negligible (in the security parameter).

To encapsulate DLog challenges in this task, we created a container class which you should use:

```
public class DLog_Challenge<IGroupElement> {
    public final IGroupElement generator;
    public final IGroupElement x;
}
```

The Reduction

Your job is to finish the implementation of the class `DLog_CDH_Reduction`:

```
public class DLog_CDH_Reduction extends A_DLog_CDH_Reduction<IGroupElement,
IRandomVariable> {

   private CDH_Challenge<IGroupElement> cdh_challenge;
   private IGroupElement generator;

   public DLog_CDH_Reduction() {
   }

   public BigInteger run(I_DLog_Challenger<IGroupElement> challenger) {
       //Implement Code here!
   }

   @Override
   public CDH_Challenge<IGroupElement> getChallenge() {
       return cdh_challenge;
   }

   private IGroupElement cdh(IGroupElement x, IGroupElement y) {
       //Implement Code here!
   }

   private IGroupElement cdh_power(IGroupElement x, BigInteger exponent) {
       //Implement Code here!
   }
}
```

`DLog_CDH_Reduction` reduces the DLog problem to the CDH problem. I.e., your job is to compute the discrete logarithm of a group element while given access to a CDH oracle. Formally, your reduction needs to solve a `DLog_Challenge`. To get this challenge, you need to ask the `I_DLog_Challenger<IGroupElement> challenger` for a challenge by calling:

```
DLog_Challenge<IGroupElement> challenge = challenger.getChallenge();
```

When you come up with a solution `BigInteger exponent` you need to `return` it in the method `run`.

For solving `challenge`, the class `DLog_CDH_Reduction` inherits a field `I_CDH_Adversary<IGroupElement> adversary;` which contains a perfect CDH Adversary:

```
public interface I_CDH_Adversary<IGroupElement> extends
IAdversary<I_CDH_Challenger<IGroupElement>, IGroupElement> {
    public IGroupElement run(I_CDH_Challenger<IGroupElement> challenger);
}
```

When calling `adversary.run(this)`, `adversary` will try to ask for a CDH challenge from your reduction. For this end, the method `getChallenge()` will be called and will return the value of the field `CDH_Challenge<IGroupElement> cdh_challenge` of your reduction.

When `adversary` can succesfully ask for a `cdh_challenge` it will try to solve `cdh_challenge`. This means, if `cdh_challenge` contains the group elements (g,g^x, g^y), `adversary` will return the group element g^{xy} in its `adversary.run(this)` call.

Now, given `adversary`, your task is to solve `DLog_Challenge<IGroupElement> challenge`. `challenge` will consist of two group elements `generator, x` which can be written as g, g^x. You need to return a BigInteger $e \in \mathbb{Z}_p$ s.t. $g^e = g^x$ (i.e. $x = e$).

BigInteger

In this task, you need to work a lot with the class `java.math.BigInteger`, which we use to represent big numbers. The class `java.math.BigInteger` is immutable i.e. the values of given instances cannot be changed. When you apply mathematical operations, new instances of `java.math.BigInteger` are generated. Be aware of the following methods of `java.math.BigInteger`:

```
package java.math;

public class BigInteger extends Number implements Comparable<BigInteger> {
    public static BigInteger valueOf(long val);

    public static final BigInteger ZERO;
    public static final BigInteger ONE;
    public static final BigInteger TWO;
    private static final BigInteger NEGATIVE_ONE;

    public BigInteger add(BigInteger val);
    public BigInteger subtract(BigInteger val);
    public BigInteger multiply(BigInteger val);
    public BigInteger divide(BigInteger val);

    public int signum();
```

```
    public BigInteger mod(BigInteger m);
    public BigInteger modPow(BigInteger exponent, BigInteger m);
    public BigInteger modInverse(BigInteger m);

    public boolean testBit(int n);
    public int bitLength();

    public boolean equals(Object x);
}
```

You will need some of those methods to solve this task.

PrimesHelper

There are four prime numbers p_1, p_2, p_3, p_4 that have the following three properties:

- $p_i \geq 2^{80}$,
- $p_i - 1 = \prod_{i = 1}^l q_i$ decomposes into a product of prime factors $q_1, \ldots, q_l$ that are all ≤ 71.
- $p_i - 1$ is *square-free* i.e. there is no prime number q_i s.t. q_i^2 divides $p_i - 1$.

To simplify the challenge of this task, you are guaranteed that the order of the group $\mathbb{G}$ we use in this task will **always** be one of the above four prime numbers. This means, in particular, that the number $\text{ord}(\mathbb{G}) - 1$ has a decomposition $$\text{ord}(\mathbb{G}) - 1 = \prod_{i = 1}^l q_i$$ into $l\leq 20$ prime numbers where each prime number occurs at most once.

To help you work with those group orders we implemented a class `PrimesHelper`:

```
public final class PrimesHelper {
    public static int[] getDecompositionOfPhi(BigInteger prime);
    public static BigInteger getGenerator(BigInteger prime);
}
```

If `order` is the order of the group, then the call `PrimesHelper.getDecompositionOfPhi(order)` will return an array of prime factors of `order - 1`. I.e., on input $\text{ord}(\mathbb{G})$, the method `getDecompositionOfPhi` will return an array $[q_1, \ldots, q_l]$ of small prime numbers s.t. $$\text{ord}(\mathbb{G}) - 1 = \prod_{i = 1}^l q_i.$$

The call `PrimesHelper.getGenerator(order)` will return a number $z \in \mathbb{Z}_p$ s.t. z generates the multiplicative group $\mathbb{Z}_p^\times$. I.e., we have $$ { 1, z, z^2 \mod p, ..., z^{p-2} \mod p } = { 1, 2, \ldots, p -1 }. $$

Use both methods of this class to solve this task.

CRTHelper

The Chinese Remainder Theorem states that, if we have coprime numbers $q_1, \ldots, q_l \in \N$, then the mapping $$\phi : \mathbb{Z}/{q_1\cdots q_l} \longrightarrow \mathbb{Z}/{q_1}\times \ldots \times

\mathbb{Z}_{q_l} $$ $$x \longmapsto (x\mod q_1, \ldots, x \mod q_l)$$ is an isomorphism of rings. While it is easy to compute ϕ, it is a bit tricky to compute its inverse. Because of this, we implemented the clas `CRTHelper` for you:

```
public final class CRTHelper {
    public static BigInteger crtCompose(int[] values, int[] moduli);
}
```

Use the method `CRTHelper.crtCompose` to compute the inverse of ϕ. When given arrays `int[] values`$ = {v_1, \ldots, v_l}$ and `int[] moduli`$ = {q_1, \ldots, q_l}$, `CRTHelper.crtCompose` will return a BigInteger z s.t. we have for each $i\in[l]$ $$z \mod q_i = v_i .$$

Complexity

An obvious solution to compute the discrete logarithm of a group element g^x is to brute-force it i.e. to check for each $x' \in \mathbb{Z}_p$ if $$ g^{x'} = g^x .$$ This 'solution' is obviously infeasible since it would need to make $p>2^{80}$ calls to the group oracle.

We expect your solution to be faster. In fact, your solution should have a runtime complexity of $O(\log p)$ by using the CDH oracle provided to your reduction.

Note, that after the submission deadline, we will test your solutions on our systems. To exclude infeasible brute-force solutions, our testing system will automatically assign zero points to solutions that need more than a few minutes for all testcases. If your solution for this task is correct, it should complete all testcases in under a minute when you test it.

Tips and Tricks

This task is hard. It is not obvious how you should use the CDH oracle to solve the DLog problem (g,g^x). Therefore, we will give you a few hints in the following:

- Right at the beginning of your code, you can distinguish two cases:
    1. If $g^x = g^0$, then x must be zero and the task is done.
    2. Otherwise, x is not zero modulo p and must lie in $\mathbb{Z}_p^\times$ (where p is the group order). By asking `PrimesHelper`, you can learn the generator z of $\mathbb{Z}_p^\times$. Now you know that there must be a number $k \in {0,\ldots, p - 2}$ s.t. $$x = z^k \mod p.$$ I.e., the problem of finding x is equivalent to the problem of finding k s.t. $g^x = g^{z^k}$.
- The multiplicative group $\mathbb{Z}*p^*times$ has very small subgroups, since $p - 1 = \prod{i = 1}^l q_i$. In fact, for every $i \in [l]$, the set $$ (\mathbb{Z}_p^\times)^{\frac{p-1}{q_i}} = { a^{\frac{p-1}{q_i}} \mod p ~|~ a \in \mathbb{Z}_p^\times } $$ is a subgroup of $\mathbb{Z}_p^\times$ of order q_i. If z generates $\mathbb{Z}_p^\times$, then $z^{\frac{p-1}{q_i}}$ generates $(\mathbb{Z}_p^\times)^{\frac{p-1}{q_i}}$. Now, if we would know that x would lie in $(\mathbb{Z}_p^\times)^{\frac{p-1}{q_i}}$, then there would be a simple way to determine x by brute-force. In fact, one can compute all $$ g^{1}, g^{z^{\frac{p-1}{q_i}}}, g^{z^{2\frac{p-1}{q_i}}},

\ldots, g^{z^{(q_i - 1)\frac{p-1}{q_i}}} $$ and check for which $k\in {0,..., q_i -1}$ one has $$ g^{z^{k\frac{p-1}{q_i}}} = g^x. $$ This would only take time in $O(q_i) = O(71)$.

- Note, that the solution file contains a template for a method `IGroupElement cdh_power(IGroupElement x, BigInteger exponent)`. When given a group element g^x and an exponent e, this method should (by using the method `cdh`) return the group element g^{x^e}. An obvious method to implement `cdh_power`, would be to use the CDH oracle to multiply g^x $e-1$-times with itself. However, this has a runtime complexity of $O(e)$ and e may be exponential. To implement this method efficiently, think of fast exponential algorithms for normal (positive) integer exponents!

- You are given the method `CRTHelper.crtCompose` and you know that $p-1$ decomposes into many different small prime numbers $q_1, \ldots, q_l$. So, it is advisable to first compute the number k in $g^x = g^{z^k}$ modulo each q_i and then to compose back the number $k \mod p - 1$.

Constructors

Do **under no circumstances** change or remove the constructor of `DLog_CDH_Reduction` which we pre-implemented. The TestRunner needs this empty constructor to test your solution. If this constructor does not exist or work, then the TestRunner can not test your solution and you will receive 0 points.

Submission

Please submit your solution on CodeExpert in the corresponding task.

Testing your implementation

The recommended technical setup uses VSCode and Docker.

- Install both on your system according to the instructions on the respective website.
- Download the container.zip file and unzip it.
- Open VSCode, press F1 and select `Extensions: Install Extensions...` and install the extension `Dev Containers` by Microsoft.
- Press F1 and select `Remote Containers: Open Folder in Container...` and select the unzipped folder `{{task}}`.
- Click "Yes, I trust the authors" (optionally also check "Trust the authors of all files on the parent folder 'build'").
- You can run/debug you solution by clicking `Run`/`Debug` directly above the `main` method of the respective `*TestRunner.java` file. (It might take a while until the java extension initialization is completed.)

Alternatively, you can also install a JDK/JRE 17 and run the code locally or on an online service like codio.

Scores and Points

If your implementation is correct, it should succeed in each test case and get full points.

Important Note: The tests which we run in Code Expert and on the TestRunner we gave you are only **preliminary**. After the submission deadline, we will run more exhaustive tests on your solution and review it manually.

Therefore, a solution which is only partially correct may receive full points in the preliminary tests but will get only partial points, eventually. Therefore, make sure that your reductions are correct in the formally theoretic sense of cryptographic reductions!

Time and Memory Restrictions

The resources the TestRunner can use to test your solution are limited. We expect your solution to use less than 1 minute of CPU time and a restriced space of memory when run several times.

Solutions which run into TimeOut- or OutOfMemoryExceptions will be rejected by us and receive 0 points.

Cheater Warning

The purpose of this task is to algorithmically reduce the DLog problem to the CDH problem.

Any solution which tries to solve the discrete logarithm problem by cryptoanalytical algorithms or by "tricking" the testing environment is considered to be a cheating attempt and will receive zero points.