A DDH based signature scheme by Katz and Wang (7 points for EUF-NMA and 10 points for EUF-CMA)

The signature scheme

Similar to Schnorr signatures, the scheme by Katz and Wang uses the Fiat-Shamir technique, but the keys consists of Diffie-Hellman tuples. We again operate in a group $\mathbb{G}$ of order p.

Key Generation

The key generation algorithm samples two group generators g, h, and a secret key x. It sets the public key $pk = (g,h, y_1 =g^x, y_2 = h^x)$ and the secret key to be x. It outputs pk,sk.

Signature Generation

To generate a signature on a message m, sample $r \gets \mathbb{Z}_p$ and compute $A = g^r$, $B = h^r$. Then compute the hash $ c= H(A,B,m)$ and $s = cx + r \mod p$. The signature is (c,s).

Verification

For verification, check that $c = H(g^s \cdot y_1^{-c}, h^s \cdot y_2^{-c}, m)$.

The Decisional Diffie-Hellman Problem (DDH)

The DDH problem is a problem over groups where an adversary is challenged to distinguish (g, g^x, g^y, g^{xy}) from (g,g^x,g^y,g^z) for randomly chosen $x, y, z \in \mathbb{Z}_p$.

In our framework, your reduction will get to interact with a DDH challenger. It can obtain a DDH challenge of the form

```
public class DDH_Challenge<IGroupElement> {
    /**
     * A generator of the group. Usually an encoding of one.
     */
    public final IGroupElement generator;
    /**
     * A group element of the form g^x, where g is the generator in this
tuple.
     * Usually, g^x was drawn uniformly random from the group.
     */
    public final IGroupElement x;
    /**
     * A group element of the form g^y, where g is the generator in this
tuple.
     * Usually, g^y was drawn uniformly random from the group.
     */
    public final IGroupElement y;
    /**
     * A group element of the form g^z, where g is the generator in this
```

```
    tuple. This
         * tuple is a real DDH tuple iff z = x * y. If this tuple is not real,
         * then g^z was drawn uniformly from the group.
         */
        public final IGroupElement z;
    }
```

from its challenger using the method `challenger.getChallenge()`. We call the tuple *real* if it is of the form (g, g^x, g^y, g^{xy}) and *random* otherwise. If your reduction thinks that the tuple was of the type *real* it should return the bit `true`, otherwise `false` as the return value of its `run` method.

EUF-NMA attacks

In the EUF-NMA setting, your reduction will be provided with an adversary that will forge a signature with a non-negligible probability when given a valid public key and access to a hash function.

Your reduction should implement the methods `public Boolean run(I_DDH_Challenger<IGroupElement, BigInteger> challenger)` which needs to solve the DDH challenge, as well as `public KatzWangPK<IGroupElement> getChallenge()` and `public BigInteger hash(IGroupElement comm1, IGroupElement comm2, String message)` which will be called by the adversary during its run to obtain the public key and hash values it needs.

You can invoke your adversary using the method `adversary.run(this)` and it may call the other methods provided by your reduction during its run.

We note that it is possible for this scheme to write a reduction that does not rewind the adversary. It is also possible to find a reduction that rewinds the adversary, but such a reduction will only be awarded at most half the possible points.

In particular, if your reduction does not rewind the adversary, we ask you to not make any calls to `adversary.reset(long seed)` as such calls will be counted as rewinds even if you do not call the `run` method after or if you have not called the `run` method before.

EUF-CMA attacks

In the EUF-CMA setting, the adversary makes chosen-message attacks, i.e. it will ask its challenger (your reduction) to sign messages of its choice.

Your reduction will need to implement the method `public KatzWangSignature<BigInteger> sign(String message)` in addition to all methods described in the NMA setting.

We note that it is still possible to write a reduction that does not rewind the adversary and only such a non-rewinding reduction will be awarded the full points. Do not reset the adversary if you do not rewind it, as a reduction that makes calls to the `reset` method will only be awarded at most half of the points.

The ideal reduction

To achieve full points, your reductions should run the adversary only once (keep in mind that calls to the `reset` function will be counted as extra runs!) and for any adversary that has success probability

ε in the EUF-NMA or EUF-CMA games, respectively, your reduction should have success probability $\frac{1 + \varepsilon}{2}$. If your reduction has a lower success probability, even just off by constant factors, you may not get full points.

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 5 minutes 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 show EUF-NMA or EUF-CMA security of the Katz-Wang scheme assuming DDH.

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