**Information Security Lab - AS21**

# Reductions

*Handout: Oct 22, 2021 12:00 AM*

*Due: Nov 8, 2021 3:00 PM*

---

## DCR to Paillier

Open Task

# Reduction of DCR to Paillier (25 points)

## The Decisional Composite Residuosity (DCR) assumption

The DCR assumption is that the DCR game is hard to win for any PPT adversary. The DCR game is defined as follows:

The **challenger** generates an RSA modulus $N = p \cdot q$ where $p$ and $q$ are two different primes with a bitlength $\lambda$ (the security parameter).

The challenger draws a bit $b$ and a value $y \leftarrow \mathbb{Z}_{N^2}$ uniformly at random.

The challenger sets

$$z := \begin{cases} y & \text{if } b = 0 \\ y^N & \text{if } b = 1 \end{cases}$$

and gives $(N, z)$ to the **adversary** who must decide whether there exists some $\hat{y} \in \mathbb{Z}_{N^2}$ s.t. $\hat{y}^N = z$.

The adversary returns a guess bit $b'$ and wins iff $b' \stackrel{?}{=} b$.

## The (simplified) Paillier Encryption Scheme (Paillier PKE)

The Paillier PKE consist of the following three algorithms:

- $\mathbf{Gen}(1^\lambda)$: Sample an RSA modulus $N = p \cdot q$ for security parameter $\lambda$. Output $\mathbf{pk} := N$ and $\mathbf{sk} := (N, \Phi(N))$ where $\Phi(N) = (p-1) \cdot (q-1)$.
- $\mathbf{Enc}(\mathbf{pk}, m \in \mathbb{Z}_N)$: Sample encryption randomness $r \leftarrow \mathbb{Z}_N^*$ and output the ciphertext $c := r^N (1+N)^m \mod N^2$.
- $\mathbf{Dec}(\mathbf{sk}, c \in \mathbb{Z}_{N^2}^*)$: Compute $w := c^{\Phi(N)} \mod N^2 \equiv r^{N\Phi(N)}(1+N)^{m\Phi(N)} \equiv (1+N)^{m\Phi(N)} \equiv 1 + m\Phi(N)N$ and output $m := (w-1)/N * (\Phi(N)^{-1} \mod N) \mod N$.

## The reduction

Your task is to produce a **reduction** that uses a provided IND-CPA adversary (with an interface as described in the lecture) for the Paillier PKE to win the DCR game. To this end you must implement the methods of the `DCR_Paillier_Reduction` class:

```
public class DCR_Paillier_Reduction extends A_DCR_Paillier_Reduction {
   // your code here
    public BigInteger getChallenge(final BigInteger m_0, final BigInteger m_1) {
        // your code here
    }
    public DCR_Modulus getPaillierPK() {
        // your code here
    }
    public Boolean run(I_DCR_Challenger challenger) {
        // your code here
    }
}
```

The running environment will start the reduction via its `run` method providing a `DCR_Challenger` from which the reduction can query a challenge via `challenger.getChallenge()`. The challenge looks as expected:

```
public class DCR_Challenge {
    public final BigInteger z;
    public final DCR_Modulus n;
}
```

To help you solve this challenge, the reduction can use the aid of an adversary for the Paillier PKE via `adversary.run(this)`. However, this adversary expects that it is able to query a Paillier public key and a challenge via the reduction's methods (oracles) `getPaillierPK` and `getChallenge` which you have to implement.

## Tightness

In this task, we are interested in **tight** reductions. This means that your reduction may call `adversary.run` at most once.

Reductions which call `adversary.run` twice or more during a run will only receive **partial** points!

## Constructors

Do **under no circumstances** change or remove the constructor of `DCR_Paillier_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.

# Testing Your Implementation

To test your implementation, you can use the Run- and Test-Button of the Code-Expert GUI. When you do this, the TestRunner will try to compile your reduction and play the DCR game (described above) several hundred times with it to estimate the advantage of your reduction.

## Scores and Points

If the measured advantage is high enough and your reduction is tight, then you should receive full points (25 of 25).

If your reduction is not tight, you will only receive partial points. If your reduction does not follow the rules which we explained here it might have a negligible advantage and will get zero points.

After each run, the TestRunner will tell you how many points your solution got in the *preliminary* tests.

**Important Note:** The tests which we run in Code Expert 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 on Code Expert 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 10 seconds of CPU time and a restricted space of memory when run several hundred 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 DCR problem to the security of the Paillier PKE.

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