TP: Side-Channel Attack Against RSA Exponentiation

Jean-Sébastien Coron

University of Luxembourg

1 Practical setup

Install SageMath (`http://www.sagemath.org/`) or use SageCell (`https://sagecell.sagemath.org`).
Please submit a single `.ipynb` file containing:

- your code,
- short comments,
- validation tests.

2 RSA decryption with square-and-multiply

We consider RSA decryption:

$$m = c^d \bmod N$$

with left-to-right square-and-multiply.

1. Implement RSA key generation:

```
def keyGen(n=512):
    """Return (Nn, p, q, e, d) with RSA modulus size about n bits."""
    return Nn, p, q, e, d
```

2. Implement decryption with the vulnerable algorithm:

```
def decrypt_vuln(c, Nn, d):
    """Square-and-multiply decryption."""
    return m
```

3. Validate correctness:

```
def checkDec():
    Nn, p, q, e, d = keyGen()
    m = ZZ.random_element(Nn)
    c = power_mod(m, e, Nn)
    assert decrypt_vuln(c, Nn, d) == m
```

3 Leakage model and attack

Assume the attacker knows (N, e) and one ciphertext c, and can observe one side-channel bit after each loop iteration:

$$\lambda_i = \mathrm{LSB}(z_i)$$

where z_i is the intermediate register value after processing bit d_i.

1. Implement a leakage oracle:

```
def leak_lsb_trace(c, Nn, d):
    """Return the sequence [lambda_i] produced during decrypt_vuln."""
    return trace
```

2. Implement a key-recovery attack that reconstructs d bit by bit (from most significant unknown bit to least significant) by:
   - forward simulation under key-prefix hypotheses,
   - consistency checks with the observed leakage trace.

```
def recover_d_from_lsb(c, Nn, e, trace):
    """Recover the private exponent d from leakage trace."""
    return d_rec
```

3. Test your attack on random keys and report:
   - success rate,
   - runtime for 256-bit and 512-bit moduli.

```
def checkAttack(trials=10, n=256):
    for _ in range(trials):
        Nn, p, q, e, d = keyGen(n)
        c = ZZ.random_element(2, Nn-1)
        tr = leak_lsb_trace(c, Nn, d)
        d_rec = recover_d_from_lsb(c, Nn, e, tr)
        assert d_rec == d
```

4 Countermeasures

Implement two countermeasures and evaluate their impact against your attack.

1. Exponent blinding:
$$d' = d + k\,\varphi(N)$$
for random k at each execution.
2. Message blinding:
$$c' = c\,r^e \bmod N, \quad m' = (c')^d \bmod N, \quad m = m'\,r^{-1} \bmod N.$$

For each countermeasure:

- implement it,
- show decryption remains correct,
- run your previous recovery attack and discuss what fails.