Lab 3: Coppersmith's attack on Low RSA exponent

1 Coppersmith's algorithm for small root finding

Coppersmith's attack on the one-way RSA function relies on the following theorem shown in [1].

Theorem 1. *Let $N \in \mathbb{N}_+$, $f \in \mathbb{Z}[x]$ be a monic polynomial of degree n. Let further $X = N^{\frac{1}{n}-\varepsilon}$ for $\varepsilon > 0$. Then there exists an algorithm that finds all $|x_0| < X$ satisfying $f(x_0) = 0 \bmod N$, in time equal to the running time of the LLL algorithm on a lattice of dimension $\mathcal{O}(\min\{\frac{1}{\varepsilon}, \log_2 N\})$.*

A nice feature of this theorem is that the modulus N can be composite with unknown factorization (for prime moduli there is no need in this theorem because we know more efficient algorithms to factor polynomials in such cases).

In order to prove Coppersmith's theorem, we start with the result due to Howgrave-Graham [2]. In what follows, with a polynomial $h(x) = \sum_{i=0}^{n} a_i x^i \in \mathbb{Z}[x]$ we associate the coefficient vector $(a_i)_i \in \mathbb{Z}^{n_1}$ and define its squared norm as $\|h\|^2 = \sum_i |a_i|^2$.

Lemma 2. Let $h(x) \in \mathbb{Z}[x]$ be a polynomial of degree n and $X > 0$ be integer. Assume, $\|h(xX)\| < N/\sqrt{n}$. If $|x_0| < X$ satisfies $h(x_0) = 0 \bmod N$, then $h(x_0) = 0$ holds over the integers.

Proof. Consider

$$\begin{aligned} |h(x_0)| &= \left|\sum_i a_i x_0^i\right| = \left|\sum_i a_i X^i \left(\frac{x_0}{X}\right)^i\right| \le \sum_i \left|a_i X^i \left(\frac{x_0}{X}\right)^i\right| \\ &< \sum_i \left|a_i X^i\right| \le \sqrt{n}\,\|h(xX)\| < N. \end{aligned}$$

From this inequality and the condition $h(x_0) = 0 \bmod N$, it follows that $h(x_0) \equiv 0$. □

Lemma 2 states that if h is a polynomial of small norm, then all its roots mod N that are small in the absolute value, are also its roots over $\mathbb{Z}$. Therefore, the idea is, for a polynomial $f(x)$ not necessarily of small norm, we will be searching for a polynomial $h(x)$ of *small norm* whose roots coincide with the roots of $f(x)$.

We could have started with linear combinations of the polynomials $f, xf, x^2 f, \ldots$, but they are unlikely to give the desired norm. Coppersmith suggested to add to the above list of polynomials powers of $f(x)$ noting that if $f(x) = 0 \bmod N$, then $f(x)^i = 0 \bmod N^i$ for any $i > 1$. Overall, define, for some integer m,[1] the following polynomials

$$g_{i,j}(x) = N^{m-i} x^j f(x)^i, \quad \text{for } i = 0, \ldots, m-1, j = 0, \ldots n-1.$$

Then an x_0 – a root of $f(x)$ – is a root of $g_{i,j}(x) \bmod N^m$ for all $i \ge 0$. Now we are ready to search for $h(x)$ – a linear combination of $g_{i,j}(x)$'s such that the norm of $h(xX)$ does not exceed N^m (such a choice of polynomials $g_{i,j}(xX)$ allows to increase the bound from N to N^m).

Now let us dive into the solving the problem of finding a small-norm linear combination of polynomials. Identifying $g_{i,j}(xX)$ with its coefficient vector, the task of finding $h(x)$ boils down to the task of finding a short vector in the lattice L generated by the matrix, which contains in its i-th column the coefficients of

[1] A thorough analysis shows that $m = \lceil \frac{1}{n\varepsilon} \rceil$ suffices, in practice we shall choose m to be a small constant.

x^i monomial of $g_{i,j}(x)$'s. We obtain a lattice of dimension $w = nm$ with a lower-triangular basis matrix (we first order $g_{i,j}(x)$'s by i and then by j). For example, for $n = 2, m = 3$ the matrix will be of the form

$$\begin{array}{c} \\ g_{0,0}(xX) \\ g_{0,1}(xX) \\ g_{1,0}(xX) \\ g_{1,1}(xX) \\ g_{2,0}(xX) \\ g_{2,1}(xX) \end{array} \begin{array}{c} \begin{array}{cccccc} x^0 & x^1 & x^2 & x^3 & x^4 & x^5 \end{array} \\ \begin{pmatrix} N^3 & & & & & \\ \star & N^3X & & & & \\ \star & \star & N^2X^2 & & & \\ \star & \star & \star & N^2X^3 & & \\ \star & \star & \star & \star & NX^4 & \\ \star & \star & \star & \star & \star & NX^5 \end{pmatrix} \end{array}$$

The $\star$ entries the coefficients of $g_{i,j}(xX)$, the empty entries are zeros. If we run LLL reduction on this basis (this basis is already give by the *rows!* just like in FPyLLL/Sage), we obtain a lattice vector v such that $\|v\| \leq 2^w \det(L)^{1/w}$. The determinant of L can be approximated as[2]:

$$\det(L) = \prod_{i=0}^{m-1} N^{(m-i)n} \prod_{j=0}^{n-1} \prod_{i=0}^{m-1} X^j X^{ni} = \prod_{i=1}^{m} N^{in} \prod_{i=0}^{nm-1} X^i =$$
$$= N^{\frac{m(m+1)n}{2}} X^{\frac{mn(mn-1)}{2}} \approx N^{\frac{m^2n}{2}} X^{\frac{m^2n^2}{2}}.$$

For the vector v (that corresponds to the polynomial $h(xX)$) returned by LLL to satisfy the bound from Lemma 2, the following inequality should hold

$$2^w \det(L)^{1/w} < \frac{N^m}{\sqrt{w}}.$$

Substituting the approximation for $\det(L)$ and ignoring small terms, the above inequality can be rewritten as

$$\det(L) \leq N^{mw} \iff X \leq N^{1/n},$$

that corresponds to the bound stated in Theorem 1 up to ε that appears due to simplifications.

2 What does it have to do with RSA?

2.1 Stereotypical messages

RSA encryption is based on the "one-way function with trapdoor" of the form $x \mapsto x^e \bmod N$ for some $e \in \mathbb{Z}_N^\star$: knowing $d = e^{-1} \bmod \phi(N)$, one can easily invert it. School-book RSA (which is, on its own, not secure) computes, for a message m, the corresponding ciphertext as $c = m^e \bmod N$.

One of the ways to make the exponentiation efficient is to choose small e (such a choice is insecure and is not used in modern implementations), e.g., $e = 3$. In this exercise you are going to convince yourself that $e = 3$ is a bad idea. For example, consider what is called "stereotypical messages" such as "your OTP is XXXX". An encryption for such message is $(S + x)^e \bmod N$, where S is the known part of the message ("your OTP is" in our example), and x is unknown. Then the ciphertext can be viewed as the polynomial $f(x) = (S + x)^e - c \bmod N$, where x is a root. If the public exponent e is small, then Coppersmith's algorithm efficiently recovers x since the dimension of the lattice L constructed as above, will be small.

[2] the factors that not explicitly mentioned, are moved into ε

2.2 Random padding

An attack on RSA encryption of related messages was proposed by Franklin and Reiter in 1996 [3]. Assume two messages m, m' are related as $m = m' + r$, where r is small (for example, if for the encryption of the ith message a so-called padding $R_i = i < 2^k$ is used in order to randomize the messages, then $c_i = (m \cdot 2^k + i \bmod N)$; such a randomization method is not secure). Then for $e = 3$, we have

$$c = m^3 \bmod N$$
$$c' = (m + r)^3 \bmod N.$$

From c, c', r, one can easily obtain m (think how, you'll need it later). What if we do not know r, but we know that it is small? Then the two ciphertexts c, c' give the following system

$$m^3 - c = 0 \bmod N$$
$$(m + r)^3 - c' = 0 \bmod N,$$

where the unknowns are m, r. Using the method of resultants (classical approach to eliminate unknowns from a system[3]), we obtain a univariate polynomial in r:

$$\mathrm{res}_m(m^3 - c, (m + r)^3 - c') = r^9 + (3c - 3c')r^6 + 3r^3(c^2 + 7cc' + c'^2) + (c - c')^3 \bmod N.$$

The obtained polynomial $f(r) = r^9 + (3c - 3c')r^6 + 3r^3(c^2 + 7cc' + c'^2) + (c - c')^3$ of degree 9 has r as its root. If $r < N^{1/9}$, we can efficiently find it using Theorem 1.

3 Task

In the file `Lab3Input.txt`, you are given an RSA public key $(N, e = 3)$ and two ciphertexts (c, c') encrypting (m, m') that are related via small r. You should find r and m, m'. In your implementation you can use $X = \lfloor 0.5 \cdot N^{1/9} \rfloor$, and when defining the polynomials $g_{i,j}$, you can take $m = 5$.

References

[1] Don Coppersmith. *Small solutions to polynomial equations, and low exponent RSA vulnerabilities.* Journal of Cryptology, 10:233–260, 1997

[2] Nick Howgrave-Graham *Finding small roots of univariate modular equations revisited.*. Cryptography and Coding, volume 1355 of Lecture Notes in Computer Science, 131–142. Springer-Verlag, 1997.

[3] Don Coppersmith, Matthew Franklin, Jacques Patarin, Michael Reiter. *Low-Exponent RSA with Related Messages* Eurocrypt'96.

[3] in the concrete exercise you'll need only the value of the resultant