Module 04: Elliptic Curve Cryptography and Lattice Cryptanalysis

Week-13: Coppersmith's Attack (40 Points)

Jan Gilcher, Kenny Paterson, and Kien Tuong Truong
jan.gilcher@inf.ethz.ch, kenny.paterson@inf.ethz.ch, kitruong@ethz.ch

December 2023

Hi all! Welcome to the last module for the semester!

This module's lab session will work slightly differently from the previous modules. Instead of splitting up students and TAs over several rooms, we will have one big lab session every week with all the TAs for this modules present. At the beginning of each lab session we will give a short presentation with helpful information. In this first lab for example, that will include how to setup the SageMath environment you'll be using to solve the labs.

The room for the lab sessions is: **ML D 28**.

This lab will also follow a Capture The Flag (CTF) approach, that you might already know from a previous module. For this you will be required to interact with our CTFd instance, where you will find the code and be able to submit flags. You have already received the credentials to for your account on the server via email (if you have not message us ASAP!). The server is available at:

`https://isl.aclabs.ethz.ch`

As with previous modules, you are also welcome to post your questions and to join ongoing discussions on the Moodle forum at the following link:

`https://moodle-app2.let.ethz.ch/mod/forum/view.php?id=911956`

1 Overview

In the lecture you have seen the theory behind Coppersmith's algorithm, and you have seen that it allows you to efficiently find small roots of polynomials mod N even when you don't know the factorization of N.

In this lab you will apply that knowledge to attack various schemes.

These tasks are fairly varied and provide you with cryptographic schemes which you can break using Coppersmith.

Attention: We ask you to implement your solutions for this module by using LLL as the basic building block. For this module, **you will be penalized** if your solution uses the `small_roots` method of SageMath polynomials. You should write a Coppersmith implementation that uses LLL to find small roots. You are **not allowed** to copy implementations found online.

2 Coppersmith's Attack (40 Points)

For each of these challenges, you need to understand the server code and try to reduce the problem at hand to the problem of finding small roots of a polynomial.

We suggest that you first implement a generic version of Coppersmith's algorithm by following the construction in the lecture and exercise session. Then, you may use it (and tweak it) for each challenge.

2.1 Padded RSA (12 Points)

This server implements an RSA encryption oracle. Upon an encryption request, a secret message will be encrypted under a key known to the server. You may ask for the encryption only once (this is a hint: you do not need more than one ciphertext). The server will pad your message and send you the ciphertext. Note that the encryption exponent is $e = 3$.

Your objective is to recover the secret message.

Your hand-in for this task is a single python file, `lab3m0.py`, that successfully interacts with our server and receives the flag. At the end it should print the flag to standard output (this should be on the last line of the output of your program). You can connect to the challenge server at `isl.aclabs.ethz.ch:50300`.

2.2 Export RSA (12 Points)

This server implements an RSA key-store. You can ask for new keys of a chosen size to be created. The keys will then be stored by the server. The only operation you may ask for is to be given an encrypted message containing one of the factors of the RSA modulus, for a key of your choice. You are not given the key that decrypts the ciphertext.

Your task is to obtain the flag by submitting a valid signature for the message *gimme the flag*, using any of the keys contained in the key-store.

Your hand-in for this task is a single python file, `lab3m1.py`, that successfully interacts with our server and receives the flag. At the end it should print the flag to standard output (this should be on the last line of the output of your program). You can connect to the challenge server at `isl.aclabs.ethz.ch:50310`.

2.3 Elliptic Curves + RSA = ♡ (16 Points)

In this last task for this lab, you will face an elliptic-curve scheme where the curve, instead of being defined over a prime field, is defined over $\mathbb{Z}_N$, where N is an RSA modulus.

Concretely, the scheme does a Diffie-Hellman key exchange between a long-term key (generated at the start by the server) and an ephemeral key (generated during encryption). The shared secret obtained is used as a keystream and XOR-ed with a secret message.

For decryption, a person who has the long-term key would re-compute the shared secret from the ephemeral public key and their own long-term private key and XOR the obtained shared secret with the ciphertext.

The following is the pseudocode for the encryption scheme. Let p, q be primes of 512 bits and $N = p \cdot q$. Let E be an elliptic curve over $\mathbb{Z}_N$ (for some pre-determined value of a and b), with a point $(x_G, y_G) = G \in E$ that serves as the base point.

```
$\mathrm{KeyGen}() \mapsto (\mathsf{sk}, \mathsf{pk})$
$\mathsf{sk} \leftarrow\!\!\$ \{1, ..., N\}$
$\mathsf{pk} \leftarrow [\mathsf{sk}]G$
return $(\mathsf{sk}, \mathsf{pk})$
```

```
$\mathrm{Enc}(\mathsf{pk}, m) \mapsto (\mathsf{pk}_{\mathrm{eph}}, c)$
$(\mathsf{sk}_{\mathrm{eph}}, \mathsf{pk}_{\mathrm{eph}}) \leftarrow\!\!\$ \mathrm{KeyGen}()$
$S \leftarrow [\mathsf{sk}_{\mathrm{eph}}]\mathsf{pk}$
$(x_S, y_S) \leftarrow S$
$K \leftarrow \mathsf{int\text{-}to\text{-}bytes}(x_S)||\mathsf{int\text{-}to\text{-}bytes}(y_S)$
$c \leftarrow m \oplus K$
return $(\mathsf{pk}_{\mathrm{eph}}, c)$
```

```
$\mathrm{Dec}(\mathsf{sk}, (\mathsf{pk}_{\mathrm{eph}}, c)) \mapsto m$
$S \leftarrow [\mathsf{sk}]\mathsf{pk}_{\mathrm{eph}}$
$(x_S, y_S) \leftarrow S$
$K \leftarrow \mathsf{int\text{-}to\text{-}bytes}(x_S)||\mathsf{int\text{-}to\text{-}bytes}(y_S)$
$m \leftarrow c \oplus \sigma$
return $m$
```

In the above, $\oplus$ denotes the XOR operation and $||$ denotes concatenation of byte strings.

A first observation is that, since $\mathbb{Z}_N$ is not a field, some operations may fail on the elliptic curve (since you have to compute inverses when e.g. adding points together). However, one can note that, if an operation were to fail, then we would be able to use the intermediate values of the operation to factor N.[1] For a well-formed RSA modulus N, the probability that an operation fails is very low, and hence you should not attempt to go down that road.

Furthermore, the problem of solving square roots over $\mathbb{Z}_N$ is assumed to be computationally infeasible if we do not know the factorization of N. If the factorization is known, then we can split the problem into two sub-problems, one over $\mathbb{Z}_p$ and one over $\mathbb{Z}_q$, and then compose the solutions using the Chinese remainder theorem. However, you are not given the factorization of N.

You are given a ciphertext. Find a way to decrypt it (without the private key!) to obtain the flag.

Your hand-in for this task is a single python file, `lab3m2.py`, that successfully interacts with our server and receives the flag. At the end it should print the flag to standard output (this should be on the last line of the output of your program). You can connect to the challenge server at `isl.aclabs.ethz.ch:50320`.

3 Hand-in and Grading

Your hand-in will consist of a single zip file, containing three folders, named `week1`, `week2`, and `week3`. Each contains your solution for the corresponding week. This weeks files will go into the `week3` folder. This folder should contain the following three files: `lab3m0.py`, `lab3m1.py`, and `lab3m2.py`.

Your final folder structure for the whole submission should look like this:

```
submission.zip
|-- week1
|   |-- ecdsa2.py
|   |-- lab1m0_2.py
|   |-- lab1m0_3.py
|   |-- lab1m1.py
|   |-- lab1m2.py
|   |-- lab1m3.py
|-- week2
|   |-- lab2m0.py
|   |-- lab2m1.py
|   |-- lab2m2.py
|-- week3
    |-- lab3m0.py
    |-- lab3m1.py
    |-- lab3m2.py
```

[1] Indeed, this is the intuition behind a factorization method known as Lenstra's factorization algorithm.

For each task you will get points as noted in the task description. You'll get 50% of the points when you successfully enter the flag on the CTF server. The remaining 50% will be awarded based on running your submissions repeatedly on our servers.