Side channel attacks: How to cheat at cryptography

# Introduction

Cryptography is useless when just theoretical. At some point it must be implemented in software or hardware. When doing so we bring it into the real world, where time, energy and people exist. This brings together many interactions that aren’t at first obvious when we reason about cryptography – we call these side channels. In this project you will be investigating how side channels can break some real-life cryptography.

Before we introduce the problems, we recommend you make sure you understand about bits, bytes and bases and a little bit of programming.

## Bits, Bytes and Bases

We’ve prepared a Bit, Bytes and Bases introduction document for you. Work through that and you should be set. You don’t have to send us the answers to the exercises, but you can if you’d like us to check your understanding.

## Programming

A little later on, the number of calculations we are going to need to do will get quite large. Therefore, it useful to get a computer to do them for you. We recommend <https://www.learnpython.org> for this but feel free to use any resources you can find and access.

We recommend you complete at least the following tutorials under “Learn the Basics”

* [Hello, World!](https://www.learnpython.org/en/Hello%2C_World%21)
* [Variables and Types](https://www.learnpython.org/en/Variables_and_Types)
* [Lists](https://www.learnpython.org/en/Lists)
* [Basic Operators](https://www.learnpython.org/en/Basic_Operators)
* [Conditions](https://www.learnpython.org/en/Conditions)
* [Loops](https://www.learnpython.org/en/Loops)
* [Functions](https://www.learnpython.org/en/Functions)

You may also want to read more about side channel attacks in general for background if you have time. There are many good resources online, but a good starting point can be found here: <https://www.comparitech.com/blog/information-security/side-channel-attack/>

# Introduction to Exercises

For each exercise, we’ve given you a program that you are trying to attack. To interact with these programs, you need to use your computer’s command terminal. For different computers this will be slightly different.

* Windows: search for a program in the start menu called “Command Prompt”
* MacOS: search for a program using spotlight called “Terminal”
* Linux: find the terminal program

For all computers this will open a text-based window where you can type commands and it will print out responses on the screen.

You will only need two commands

* ls (dir on Windows) will list the files and folders in the current folder
* cd will change to another directory

Use these to find the files you have downloaded.

### Running our super secure programs

Type the name of the program (with “./” before it on Mac and Linux), when you’re in the same directory as it, will run it.

* + Windows: oracle.windows
  + Linux: ./oracle.windows
  + MacOS: ./oracle.darwin
  + (Let us know if none of these work so we can make you one that will)

### Running your exercise code

For some exercises you’ll need to write some Python code to interact with it for you faster. We’d recommend [VSCode](https://code.visualstudio.com/) as a helpful code editor (particularly with its Python extension), however, any text editor works (but not Microsoft Word).

To run the python code you will need python interpreter program installed. This should be installed on the school computers. To install it on your own computer: <https://www.python.org/downloads>

To run a python program type python exercise.py

Now, find the exercise program folder and you can get going.

# Error Messages

*Aim: Understand how to break down exponential-sized searches to crack passwords*

## Exercise

The first program can be unlocked using a 4 digit pin. That’s not very many options for a computer to try but is quite tedious to do by hand. How many is it?

Try interacting with the program. See if you can come up with a quicker solution?

## Exercise

The second program is also unlocked using a pin, but with an unknown length. How many options would we have to check?

Try interacting with the program. See if you can come up with a solution?

## Exercise

The third program is unlocked using a password (lowercase + uppercase letters and spaces). If I tell you the password is less than or equal to length x, how many options might you have to check?

Try interacting with the program…this might take some time…maybe we can use our computer to help.

Try writing a program (using the given skeleton program) that works out the password quickly.

## Follow-up questions

1. Can you find any examples of real-life error message attacks? This is something you may want to cover in your end-of-project presentation.
2. What mitigations can you introduce to stop this being possible?
3. Have you noticed these mitigations?

# Timing attacks

Timing attacks use the differences in how long a program takes to help discover secret data. It’s an idea that makes more sense the more examples you see, so we suggest heading on into the exercises (with some hints also supplied in the RSA Workbook), but the general side channels resources from earlier will give some helpful examples too if you need!

Our tech team has learned about the dangers of error messages now. So, they’ve re-written the password checker as to not leak information about what went wrong. Have they fixed it now?

## Exercise

Take a look at the public.py file for exercise 4.1. Is there any timing difference you could take advantage of?

Try writing a program (using the given skeleton program) that works out the password.

## RSA

*Aim: Find out about RSA and how it can use square and multiply to calculate powers*

*Aim:* *Create simple statical experiments to break RSA using a timing attack*

Now, instead of trying to guess a password, we are going to do some more maths. RSA is a way to encrypt information that is used today all over the world. Before we look at attacking it, we need to figure out how it works. We’ve prepared an introduction document “RSA Workbook” for you to work through, but you are encouraged to do some more reading too.

## Exercise

Take a look at the public.py file for exercise 4.3. It implements an RSA decryption, however, my tech geeks think they’ve found a clever way to speed it up. I’m worried they’ve introduced a timing attack. Can you check if it’s possible to work out the secret decryption exponent “d”?

A good place to start is to implement the check function so you know when you have the right “d”.

We’ve included some test numbers in test-numbers.txt which you can use to develop your program. Remove line 10 to test out your solution on the real secret numbers.

This one is quite tricky, so we’ve got some hints for you if you need them.

First hint: Lbh unir gb erpbire bar ovg bs q ng n gvzr. (encrypted with [rot13](https://rot13.com/))

Note: it’s a similar idea to the attacks you’ll find online but not quite the same.

The bigger keys sizes do take a bit longer to run. To help we can give you some pre-recorded timings. Let us know if you want them.

When you think you have the secret number “d” you can decrypt the secret message inside ciphertext.txt

## Follow-up questions

1. Can you find any examples of real-life error message attacks? This is something you may want to cover in your end-of-project presentation.
2. What mitigations can you introduce to stop this being possible?

# Extension

*Aim: Learn about AES*

*Aim: Create some more complicated statistical experiments to break a bit of AES by power trace analysis*

Let us know if you’d like an extension task – we’ve put the aims here to give an idea of what it includes.