Introduction to Cybersecurity Elisabeth Oswald

Historical Ciphers, Security Notions

2023

Overview

Some exercises are based on paper and pencil. Unless you use a tablet (with a pencil) you must bring actual paper and pencil along.

For some parts of this lab you need to program in Python, and we expect that you have a reasonable level of proficiency in Python already. We also expect that you have Python installed on your system. You will also need the Cryptography library as well as PyCryptoDome.

We do not expect that you can finish all exercises within the 3hr lab session. We also do not require that you do all exercises. The exercises are design to help you finding the answers to the corresponding Moodle quiz.

What you should get out of this.

Today's session has one overarching aim, which is for you to get more intution about modern notions of security, in particular, the notion of indistinguishability. To get there, we will look at some historic ciphers and investigate their security against brute-force key recovery adversaries. We will also look at the one-time pad based on a one-time shift cipher and consider the notion of unconditional security. Finally you get to play the indistiguishability game.

Assessment

There is no required submission for this lab, but doing the tasks will enable you to answer questions in a small Moodle quiz which is due at the end of next week.

Paper and Pencil Exercises

Exercise 1:

What is the message embedded in the following text? What is the technique that was used to disguise the message?

```
Dear George,
Greetings to all at Bristol. Many thanks for your
letter and for the Summer examination package.
All entry forms and fees forms should be ready
for final dispatch to the syndicate by Friday
20th or at the very latest, I'm told, by the 21st.
Admin has improved here, though there's room
for improvement still; just give us all two or three
more years and we'll really show you! Please
don't let these wretched 16+ proposals destroy
your basic O and A pattern. Certainly this
sort of change, if implemented immediately,
would bring chaos.
               Sincerely Yours,
                    A.N. Other
```

Exercise 2:

Consider the following 'toy' ciphers from plaintexts $\{a, b, c, d\}$ to messages $\{w, x, y, z\}$ using keys $\{k_1, k_2, k_3, k_4\}$. Encrypt the message "abc" under different keys, and jot down the ciphertexts. Assuming that the probability for any plaintext letter is 1/4, and the probability for any key is 1/4, what is the the likelihood, upon seeing the ciphertext letter x that the plaintext letter y was encrypted? Now consider if these encryption schemes are "good"?

1.

	a	b	c	d
k_1	w	x	y	z
k_2	z	y	z	x
k_3	x	z	w	y
k_4	x	y	x	w

2.

	a	b	c	d
k_1	w	x	y	z
k_2	x	y	z	w
k_3	y	z	w	x
k_4	z	y	x	w

Explain your answer.

Exercise 3:

Consider the following extension to the shift cipher: the key is a pair of numbers a, b which must be smaller than a modulus q. The encryption function is such that a plaintext letter m is understood as an integer between in $[0, q-1]$, and the encryption rule is $c = a \cdot m + b \pmod{q}$.

Write down the encryption scheme more formally, by considering possible restrictions on a, b, as well as defining a decryption function. Show correctness of the scheme. How large is the key space if $q = 26$ (Latin alphabet) and if $q = 30$ (German alphabet)?

Exercise 4:

Consider the following situation: you work for your home country's secret service, and a ciphertext was intercepted that reads `XYAIX`. It is known that the ciphertext is due to a polyalphabetic substitution cipher using a five letter key word. A brute force search over many five letter keywords led to the possible plaintexts $RIVER$ and $FIVER$ (the brute force search got stopped because it was taking up too many resources). Before continuing further with the expensive brute force key search, you are asked to estimate how long a ciphertext would need to be before it is possible to **uniquely** determine the key (and thus plaintext). You are a bit lost by this task, and ask your Cryptography professor for help, who recommends to read Shannon's 1949 paper "Communication Theory of Secrecy Systems". After reading this paper, can you solve this task?

Exercise 5:

Suppose the 26 keys in the Shift cipher are used with equal probability, and that every letter in the plaintext is encrypted by freshly sampling a random key. Argue that for any

plaintext distribution, the one-time Shift cipher has perfect secrecy. Your argument can be based on observations from some examples (that you create), or your argument can use (more or less) formal mathematics.

Programming Exercises

Exercise 6: (Brute Force Key Search)

Write a Python function that breaks the encryption of a cipher and recovers the plaintext using a brute force search. Apply the function to a shift cipher and a substitution cipher (you may use the provided Python functions for encryption).

Estimate how long it would take (assuming doing this in Python) to break the encryption of a shift and resp. substitution cipher via a brute force search.

Exercise 7: (One time pad with 26 letter alphabet)

This exercise ties in with the previous paper and pencil exercise: you are now tasked to implement such a one-time shift cipher in Python, you may use the script "onetime.py".

Get one-time encryption running. It implements the two functions `encrypt_letter()` and `encrypt_message()`. It further defines a single plaintext and a single key for testing purposes. The script has two problems. The first problem is with the supplied test data. The second problem is in the encryption itself. Fix both problems to get the script running.

Write a decrypt function for the one-time pad with the 26 letter alphabet The principle of decryption based on the available one-time pad encryption is simple: reverse the action of encrypting by essentially "shifting" in the opposite direction: we shift each letter k positions to the left (encryption implied a right shift). This can be translated into mathematics: it implies that we compute $c = m - k \pmod{26}$ (assuming an alphabet with 26 symbols).

Write a function "decrypt_message(ciphertext, key)" that calls a function "decrypt_letter(letter, key)", which implements decryption for a shift cipher.

Observe perfect secrecy in action

Let's illustrate the meaning of "learning nothing about the plaintext". For this you will need the word list "words_alpha.txt", which I downloaded from `https://github.com/dwyl/english-words/`. This is just one example of a public word list. You can load and extract all English words using the following load function:

```
1
2 def load_words():
3     with open('words_alpha.txt') as word_file:
4         valid_words = set(word_file.read().split())
5
6     return valid_words
```

The snippet of code opens the file with the name "words_alpha.txt": because we give no further information, Python assumes it contains ASCII formated strings. We open it as object with name `word_file`, and then split up the data in this object via the `read` and `split` functions, and then "reassemble" the resulting objets as another list via the `set` function.

Add this snippet of code to your source and then call it to extract all English words via

```
1 english_words = load_words()
```

Next, using list comprehension, we will extract the list of all words with exactly three letters. Assuming you extract all English words from the word list into a list `english_words` you should create a new list, that only keeps the words with exactly length three:

```
1 three_letter_words = [word for word in english_words if len(word) == 3]
```

(This statement iterates through all the elements, which I gave the name `word`, in the list `english_words` and writes them in a new list if their length equals exactly three.)

Now that you have some code snippets that use list comprehension it is over to you! Create a list of all keys with three letters borrowing the ideas as in the given code snippets. You will want to also use the `product` function from the itertool package.

Hints: add `from itertools import product` at the beginning of your file; you can join strings via `"".join(stringname)`

With the list of all possible keys consisting of three letters, produce all possible plaintexts given the ciphertext "abc", and check how many of these plaintexts are contained in the word list. You can do this easily using the Python concept of a set intersection. The `set` class is documented here: `https://docs.python.org/3/library/stdtypes.`

`html?highlight=set#set` and it is possible to intersect two sets via `set(listA) & set(listB)`.

Exercise 8: (Win the indistinguishability game)

I explained that there are several mathematical definitions for the notion of "perfect secrecy", one of which is actually give by a game called "perfect indistinguishability". The idea is that if nothing is learned about a plaintext when given a ciphertext, then it should equally be impossible to distinguish between the encryptions of two different messages m_0 and m_1.

We call this game **perfect** indistinguishability game initially because it is played by an adversary with unlimited computational power. I also explained that in order to construct a cryptographic scheme that gives perfect secrecy = indistiguishability, it is necessary to choose a key at random and with the same length as the message. This is exactly the one-time pad, and leads to the problem of the distribution of this one-time key. Thus in some ways we have solved one problem (secure encryption) but created a new equally hard problem (secure one-time key distribution).

One idea to create then practically secure symmetric encryption is to no longer insist on security against a computationally unlimited adversary. This is what brings us to the "normal" indistinguishability game. If we drop the requirement to be secure against computationally unlimited adversaries, and "just" look at computationally limited adversaries, we can hope to be able to construct symmetric encryption schemes, by finding a construction that enable the encryption of blocks of messages re-using one secret key.

This task is now about playing the "normal" indistinguishability game in a concrete example. To do so, you must install a crypto library. There are several libraries available, and we will start with one called `PyCryptoDome`. Install this library using `pip install pycryptodome` in the terminal window of PyCharm.

Once you have installed the library you can run the script "IndGame.py". Upon running the script, the IND game is executed. You have to step into the shoes of the adversary and guess/work out which of the two displayed messages has been encrypted and resulted in the displayed ciphertext. You should notice that you can't win this game with a high probability without cheating.

Now check out the code and write a function to replace the naive adversary with a clever

adversary who formats the two messages $m0$ and $m1$ in such a way that the game can be easily won.