MysteryTwister

THE CRYPTO CHALLENGE CONTEST

Bigram Substitution – Part 1 (Tutorial)

Author: Lillin Modi + BE

September 2025

Bigram Substitution – Part 1 (Tutorial)

This is a challenge with very detailed explanations, so you can learn a lot about substitution ciphers. You will be given the tools to find the solution directly. However, you will learn the most if you program such a tool yourself.

Simple monoalphabetic substitution ciphers

Simple monoalphabetic substitution ciphers (MASC) replace every element (e.g., letters) in the plaintext (PT) with exactly one element from the ciphertext (CT). The plaintext alphabet contains all elements that can appear in the PT in an ordered fashion. In the ciphertext alphabet, these elements appear in a different arrangement (permutation), so that each element in the PT alphabet is assigned to exactly one element in the CT alphabet, and both alphabets contain all elements. This can also be seen mathematically as a bijective mapping.

For the PT alphabet of length 26 | A | B | C | … | X | Y | Z |

there are 26! possible CT alphabets, e.g., | B | X | Z | … | Y | C | A |

The CT alphabet is then the key for this encryption method.

To apply the method by hand, both alphabets are often written directly below each other.

A	B	C	…	X	Y	Z
B	X	Z	…	Y	C	A

In this table, during encryption, the letter B in the plaintext is mapped to the letter X in the ciphertext.

For an alphabet of length n, there are $n!$ different permutations. This number is very large even for an alphabet of length 26: $26! \approx 4 \times 10^{26} \approx 2^{88}$

To calculate such numbers, you can use the factorial function from Python's `math` module and output the value in scientific notation by first converting it to a string and float:

```
import math
math.factorial(3)
# yields 6
print("{:e}".format(math.factorial(26)))
# yields '4.032915e+26'
```

For very large numbers with more than 300 digits, the above conversion doesn't work, so you use the `decimal` package:

```python
import math
import decimal
c=math.factorial(576)
dc = decimal.Decimal(c)
print(format(dc, '.2e'))
# yields 4.26e+1341
print("{:.2e}".format(dc))
# yields '4.26e+1341'
```

The number of different keys for a method is also referred to as the **key space** of the method. You can find many more details on this in the CrypTool book [1], Chapter 1.6, page 8 ff, Table 1.1.

However, a large key space is only a necessary condition for a usable encryption method. Other requirements are resistance to attacks where either only the CT is known (ciphertext-only attack, COA) or where one or more CT-PT pairs are known (known-plaintext attack, KPA).

If the attacker only has the ciphertext and tries to crack a cipher, this means they want to either find the PT or determine the key, or both. If the attacker has a pair of corresponding plaintext and ciphertext, this means they want to find the key.

Both types of attacks against a simple monoalphabetic substitution are easy if the alphabet is only 26 characters long and the plaintext is normal language. If you have a much longer alphabet, only the known-plaintext attack (KPA) is still very simple.

Simple monoalphabetic substitution ciphers with long alphabets

Instead of building the PT alphabet from single letters (1-grams) as above, you can also build it from all combinations of double-letters (2-grams). The PT alphabet then looks like this:

AA | AB | AC | … | ZX | ZY | ZZ |

With 26 different single letters, there are $26^2 = 676$ different letter pairs. This means the PT alphabet has a length of 676. With the number of permutations, you can calculate how many possibilities there are for the CT alphabet. For n=676 elements, this number is n! (n factorial), a number with over 1600 decimal places (see the following Python code).

```python
c=math.factorial(676)
dc = decimal.Decimal(c)
print("{:.2e}".format(dc))
# yields 1.88e+1621
```

A specific permutation or a specific definition of the CT alphabet is then our key. In the following, we have used a randomly ordered alphabet of bigrams for the challenge.

Challenge

For this challenge, the bigram substitution was used as cipher. You receive the PT and the CT in two files:

- bgs_plaintext.txt
- bgs_ciphertext.txt

With these, you can then perform a KPA.

The idea for the plaintext comes from the novel "Post Mortem" by David Lagercrantz, (c) 2025, in which the song "Take This Waltz" by Leonard Cohen plays an important role.

Your task is to find the key and submit it in a single-line representation (only the CT alphabet). If you cannot assign a PT bigram to a CT bigram, write `??`.

The solution could look like this: `DE GH IK ZR ?? HG ?? ?? SI PQ …`

It doesn't matter whether you use a space, a tab, or a comma as a separator.

Please check before submission that you list 676 bigram pairs, including the `??`, and that apart from `??`, no bigram appears twice.

Tools

Attached are two Python programs for the bigram substitution cipher:

- `2-gram-subst.py` Tasks: gen key, enc, dec, do a simple KPA
- `test-genkey.py` Tasks: generate three keys: identity, reverse, random

With the following command in a terminal, you can see the options for the "kpa" subcommand:

```
python 2-gram-subst.py kpa --help
```

Possible solution path with the provided tool:

The following call attempts to determine the key from a matching pair of PT and CT:

```
python 2-gram-subst.py kpa --plaintext bgs_cleanedPT.txt --ciphertext
   bgs_ciphertext.txt
```

If you also add the options `--outkey` and `--cols 0`, you will get a 2028-byte file that displays the found key in a single line: `FI KC VH SQ ?? RO QX ?? YG ?? LE ... WK ?? ?? BY ?? ?? ??`

Outlook on further challenges in the bigram substitution series

This challenge was about understanding the encryption method and possibly the provided Python tools – in case you don't write your own tools.

In subsequent parts of the series:

- The tools for the solution will no longer be provided, but you now have a good foundation.
- The plaintext will be incomplete (partial known-plaintext attack, partial-KPA) or only the ciphertext will be given (COA).

If you are already interested in how long the ciphertexts must be for ciphertext-only attacks (COA) on this cipher, here are a few references (mostly by Schmeh/Dunin/van Eycke/Helm):

- https://dspace.ut.ee/items/806903ff-2f33-4676-a1be-5664bdb660b3
- https://dspace.ut.ee/server/api/core/bitstreams/c2e6dc31-5d2a-4245-949c-e26aa2a514be/content
- https://scienceblogs.de/klausis-krypto-kolumne/2017/02/13/bigram-substitution-an-old-and-simple-encryption-algorithm-that-is-hard-to-break/

Attached files for the solvers

- `2-gram-subst.py` Python program for the bigram substitution cipher (encryption and decryption, simple KPA, utils)
- `test-genkey.py` Small test program that generates the example keys below using the first Python program
- `bgs_plaintext.txt` Original plaintext
- `bgs_cleanedPT.txt` Plaintext file (stripped down to alphabet characters). This file was encrypted
- `bgs_ciphertext.txt` Ciphertext file (result of the encryption)
- `id.txt` Example key file that maps everything to itself
- `rev.txt` Example key file that maps the first bigram to the last
- `rnd.txt` Example key file with a random permutation

References

[1] Esslinger, B.: *Learning and experiencing cryptography with CrypTool and SageMath* (Artech House, Norwood, 2024).