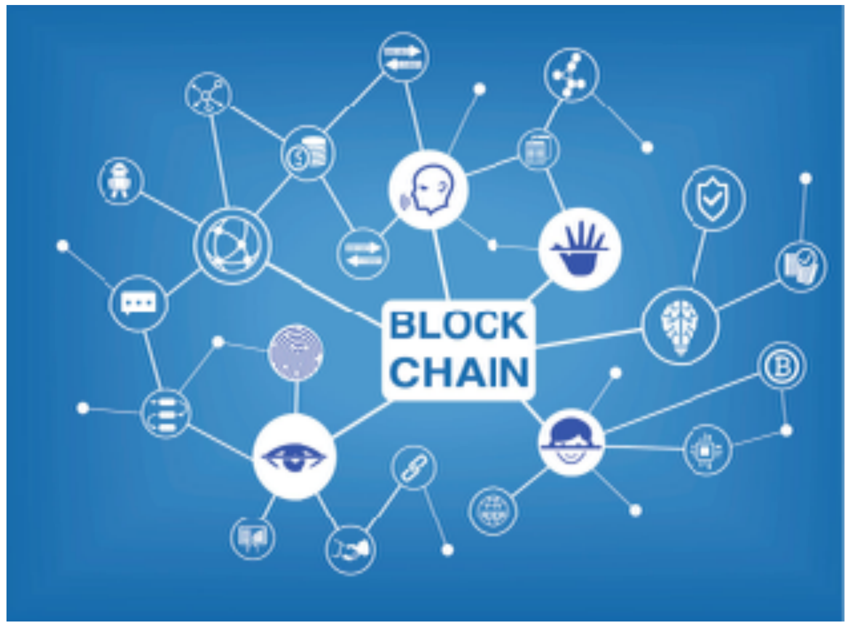
Encryption and Decryption with Python

by: Sarah Nooravi and Bat Sukhbaatar

Motivation

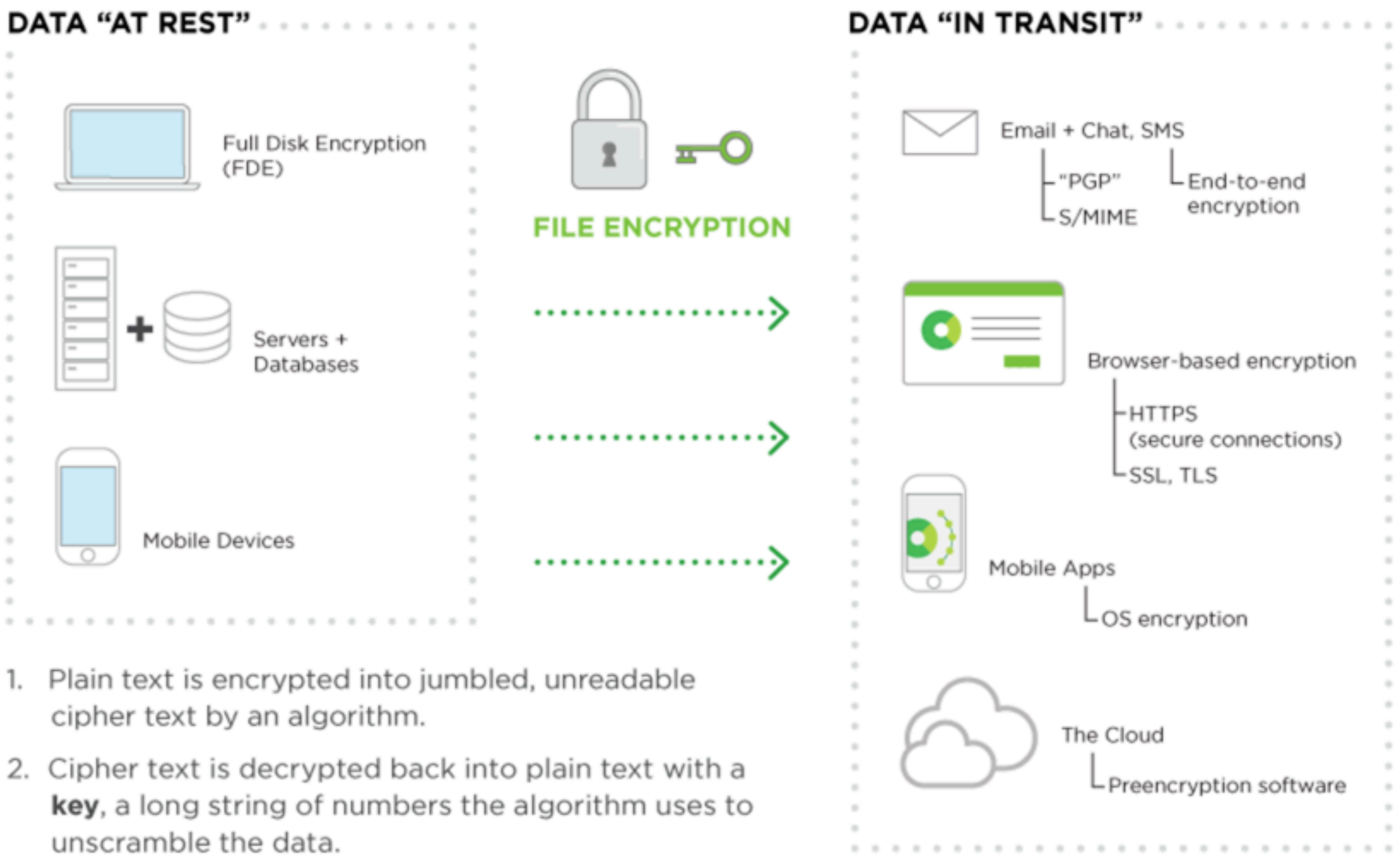
Wartime code-cracking machines such as Colossus broke German encryption systems



Encryption helps to ensure that credit card transactions stay secure

http://www.bbc.com/news/technology-24667834

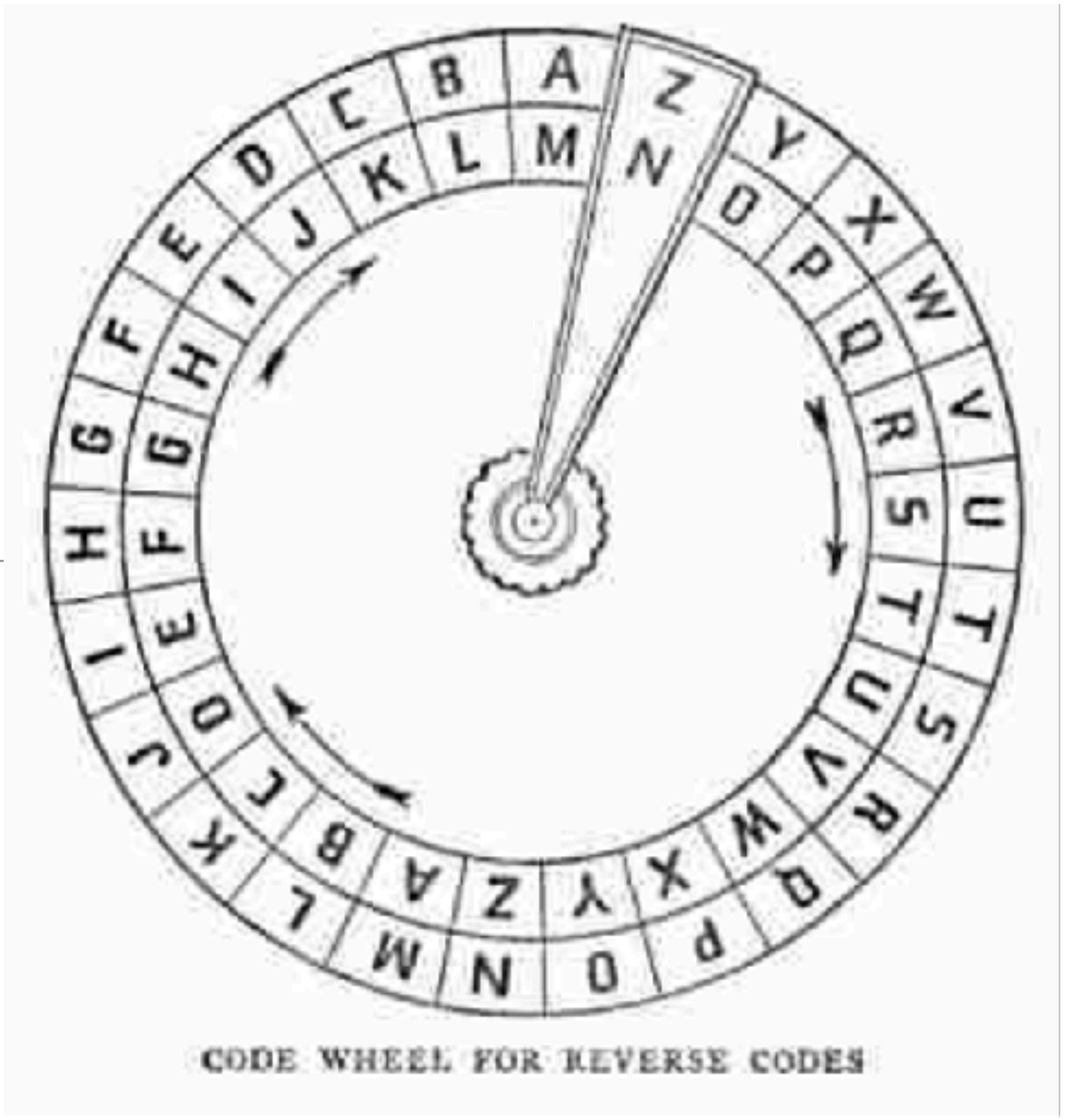
Motivation



1. Plain text is encrypted into jumbled, unreadable cipher text by an algorithm.
2. Cipher text is decrypted back into plain text with a **key**, a long string of numbers the algorithm uses to unscramble the data.

https://www.upwork.com/hiring/development/introduction-to-encryption-data-security/

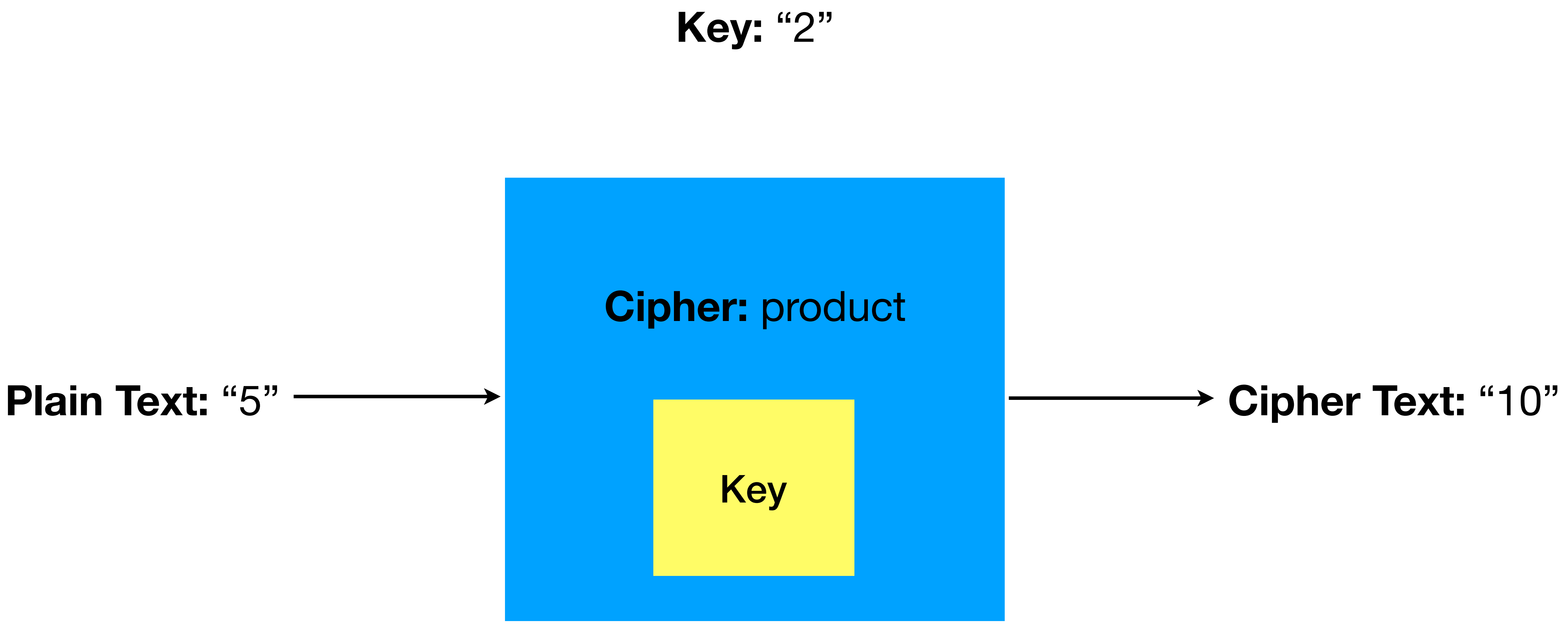
Encryption



Definitions

- **Encryption** : transforming intelligible numbers or text, sounds and images into a stream of nonsense
- **Plain text** : the original text
- **Cipher text** : the encrypted text
- **Ciphers** : the algorithms that perform the encryption and decryption
- **Keys** : a piece of auxiliary information that is used by the cipher to encode/decode the text, usually a cipher contains one or two keys
- **Decryption**: the process of taking encoded or encrypted text or other data and converting it back into text that you or the computer can read and understand.

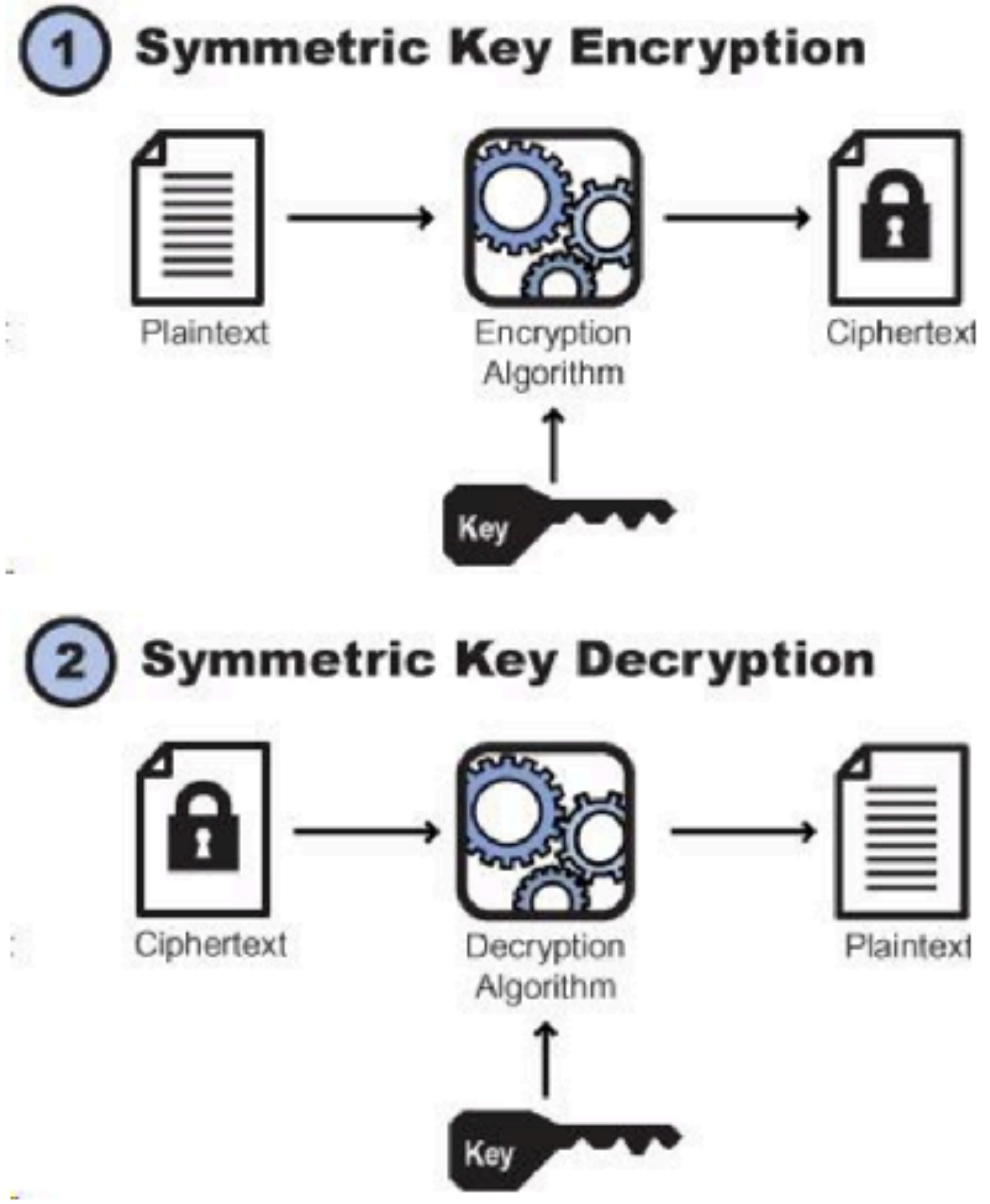
Key vs. Cipher



Definitions

Secret-Key Algorithm

- In symmetric algorithms, the sender uses a shared secret key to encrypt the message. The recipient must then use the same shared secret key to decrypt the message. Symmetric cryptography is very fast, but requires every possible pair of users to have a unique key. In a large system, the number of keys required can very quickly become unmanageable.

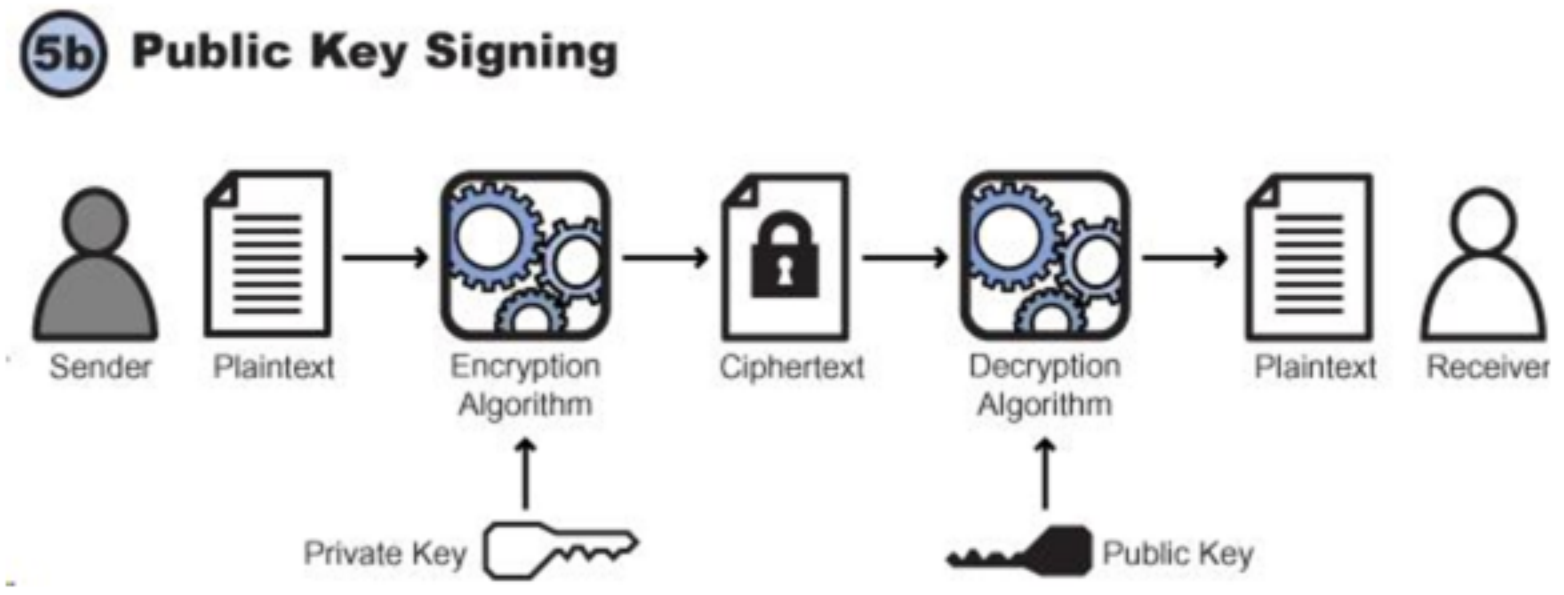


https://www.upwork.com/hiring/development/introduction-to-encryption-data-security/
http://www.infosectoday.com/Articles/Intro_to_Cryptography/Introduction_Encryption_Algorithms.htm

Definitions

Public-Key Algorithm

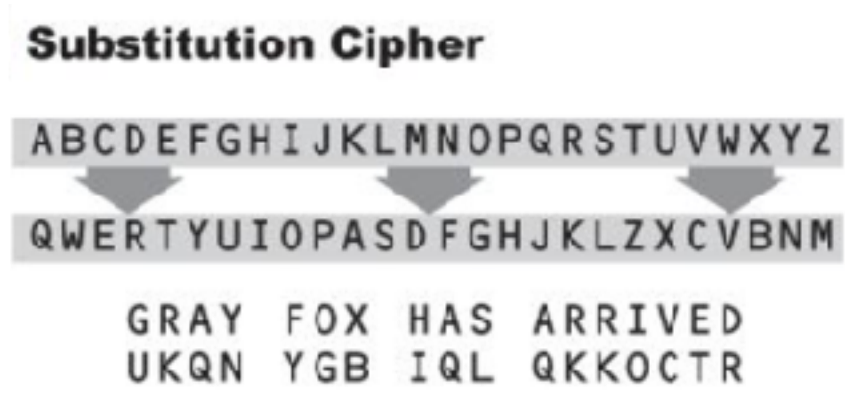
- In asymmetric algorithms (also known a public-key cryptography), the sender and recipient use related but different keys to encrypt and decrypt the message. Each participant has a pair of keys (one public and one private). and everyone has access to everyone else's public key. When a sender wants to encrypt a message, he or she does so using the recipient's public key. The recipient can then decrypt that message using his or her private key. Asymmetric cryptography is slower than symmetric cryptography, but key management is greatly simplified.



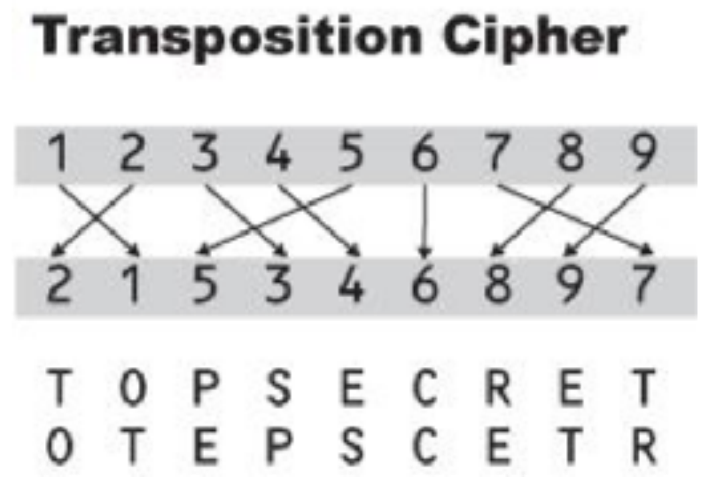
https://www.upwork.com/hiring/development/introduction-to-encryption-data-security/

Ciphers

- **Simple Substitution Ciphers**: can be demonstrated by writing the alphabet in some order to represent the substitution
  - 25! = 1.44e+25
- **Homophonic Substitution Cipher**: in an attempt to increase the difficulty of deciphering substitution ciphers, in homophonic ciphers, plaintext letters map to more than one ciphertext symbol
- **Transposition Cipher**: the units of plaintext are rearranged in a different and quite complex manner.



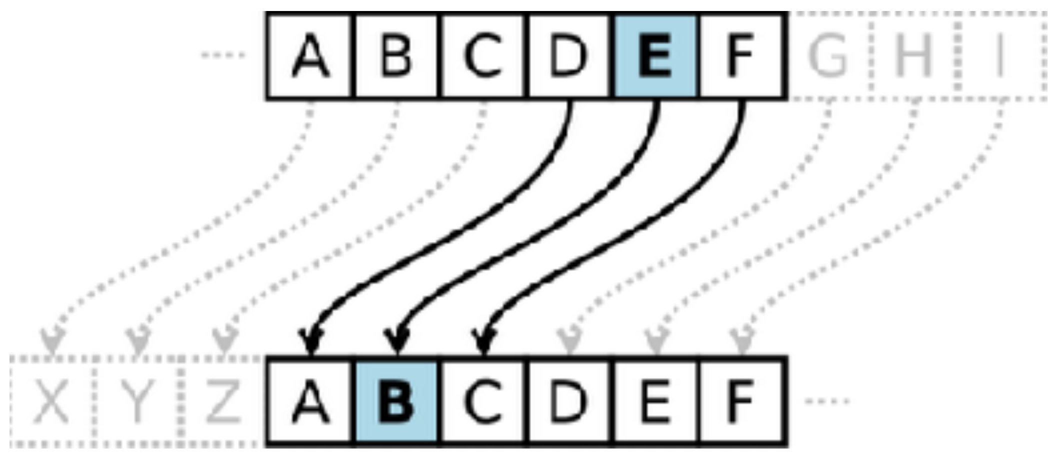
ni
ní
nì
ni4
ni5

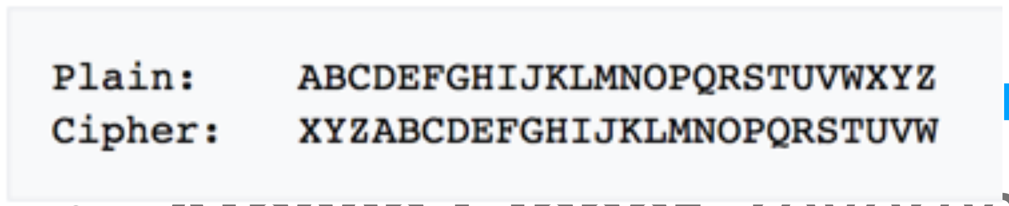


https://en.wikipedia.org/wiki/Substitution_cipher

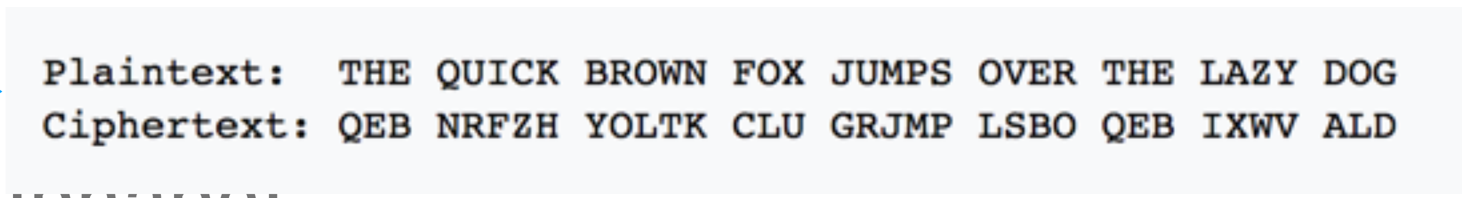
Ciphers - Examples

- **Caesar Cipher:** shifted alphabet
  - Example: a left shift of three



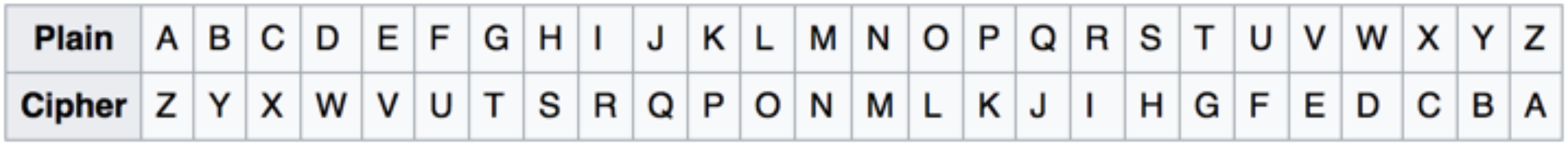


```
Plain:    ABCDEFGHIJKLMNOPQRSTUVWXYZ
Cipher:   XYZABCDEFGHIJKLMNOPQRSTUVW
```



```
Plaintext:  THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG
Ciphertext: QEB NRFZH YOLTK CLU GRJMP LSBO QEB IXWV ALD
```

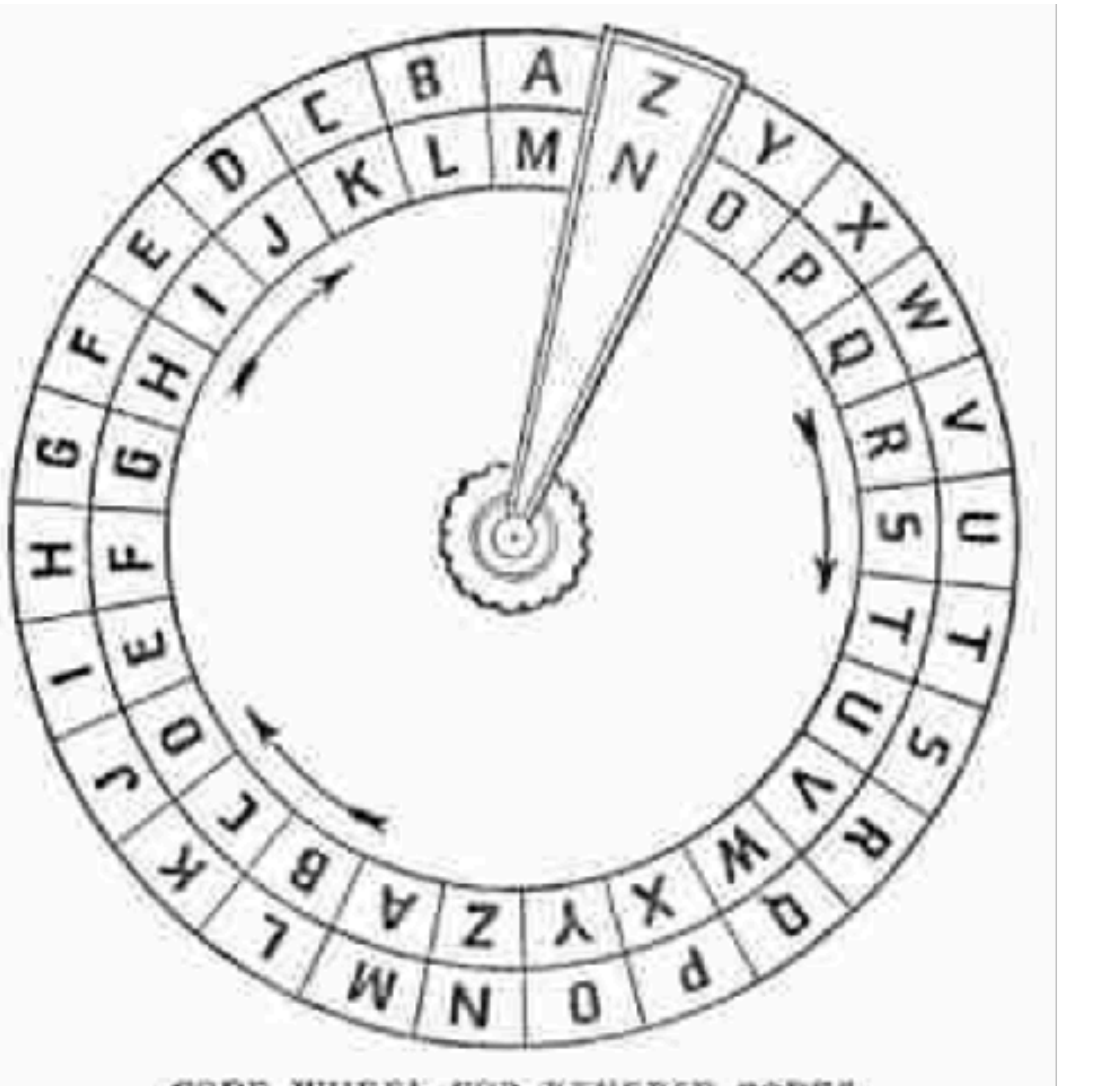
- **Atbash Cipher:** reversed alphabet



Plain	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Cipher	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A

https://en.wikipedia.org/wiki/Substitution_cipher

Decryption



CODE WHEEL FOR REVERSE CODES

CAUTION
MATH
AHEAD

Deciphering Steps

1. Create a Transition Probability Matrix on Plain Text
2. Define the Log-Likelihood loss function
3. Decipher using the Metropolis Algorithm

Step 1 - Transition Probability Matrix

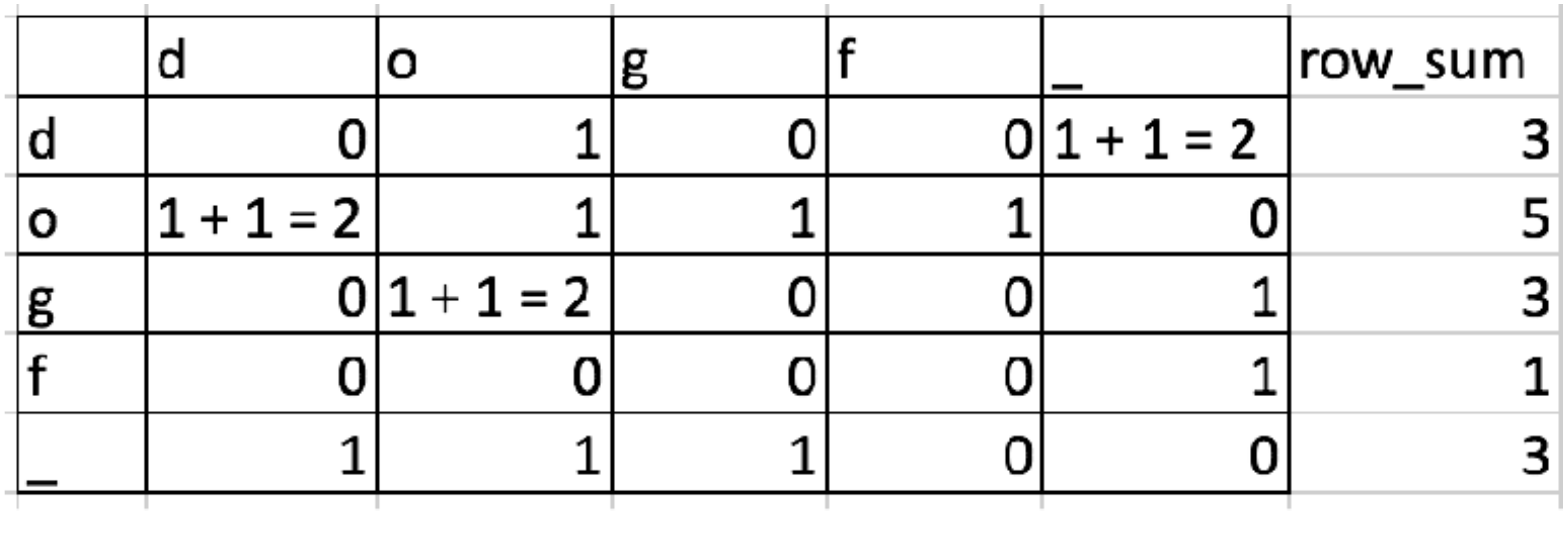
- **Definition**:
  - A square matrix used to describe the transitions of a Markov chain
- **In our case**:
  - Which letter is likely to follow which letter in English
  - 26x26 (25 English Letter, '_' for non-letter(?!+-))
- **Training**:
  - Very long English Text (War and Peace by Tolstoy)

Step 1 - Transition Probability Matrix - Example

- **English Beta:**
  - Imagine we have a simplified version of the English alphabet consisting of just 4 letters: **d, o, g, and f**
- **Example phrase in English Beta:**
  - ‘Good Dog of God.’

Step 1 - Transition Probability Matrix - Example

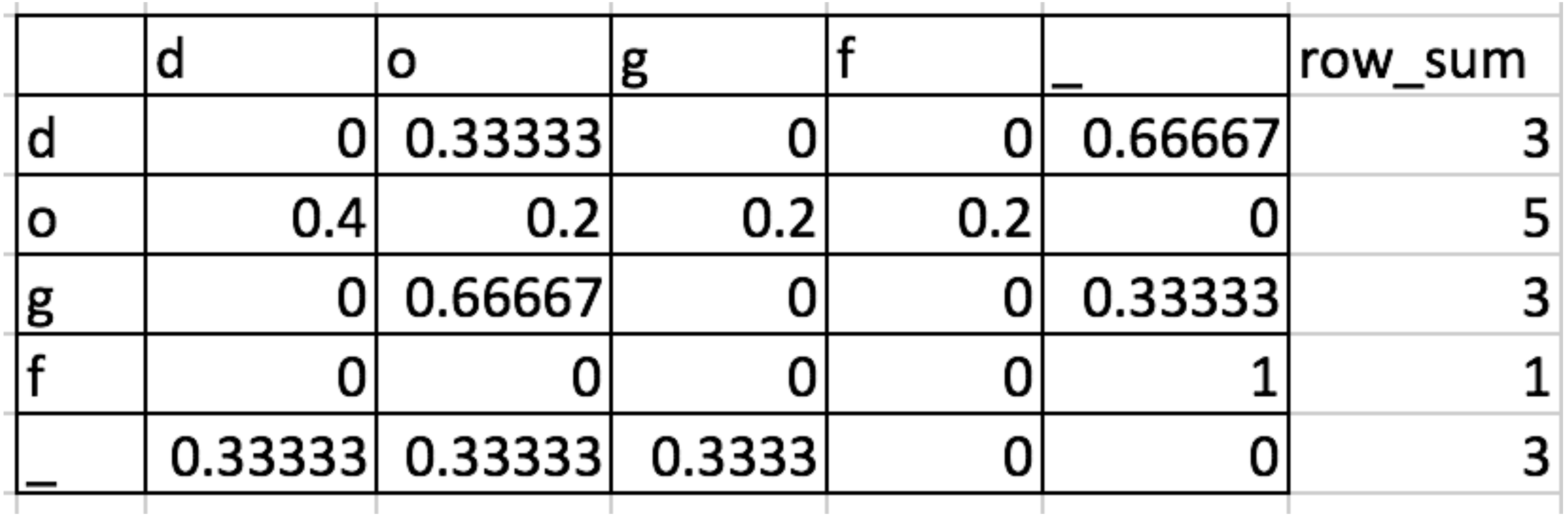
- Constructing Transition Matrix for the phrase:
  - ’Good Dog of God.’
- Loop through each two consecutive characters: ‘g’<’o’, ‘o’<’o’, ‘o'<’d’, ‘d’<’ ’, ‘ ’<’d’, ‘d’<’o’, ‘o’<‘g’, ‘g’<‘ ’, ‘ ’<‘o’, ‘o’<‘f’, ‘f’<‘ ’, ‘ ’<‘g’, ‘g’<‘o’, ‘o’<‘d’, ‘d’<‘.’



	d	o	g	f	_	row_sum
d	0	1	0	0	1 + 1 = 2	3
o	1 + 1 = 2	1	1	1	0	5
g	0	1 + 1 = 2	0	0	1	3
f	0	0	0	0	1	1
_	1	1	1	0	0	3

Step 1 - Transition Probability Matrix - Example

- Normalize by Row Sum to make probability
- Example:
  - ’o’ is likely to be followed by letter 'd' (40%)
- 5 times 'o' occurred in ex. 2 times 'o' followed by 'd'



	d	o	g	f	_	row_sum
d	0	0.33333	0	0	0.66667	3
o	0.4	0.2	0.2	0.2	0	5
g	0	0.66667	0	0	0.33333	3
f	0	0	0	0	1	1
_	0.33333	0.33333	0.3333	0	0	3

Step 2 - Defining Log Likelihood

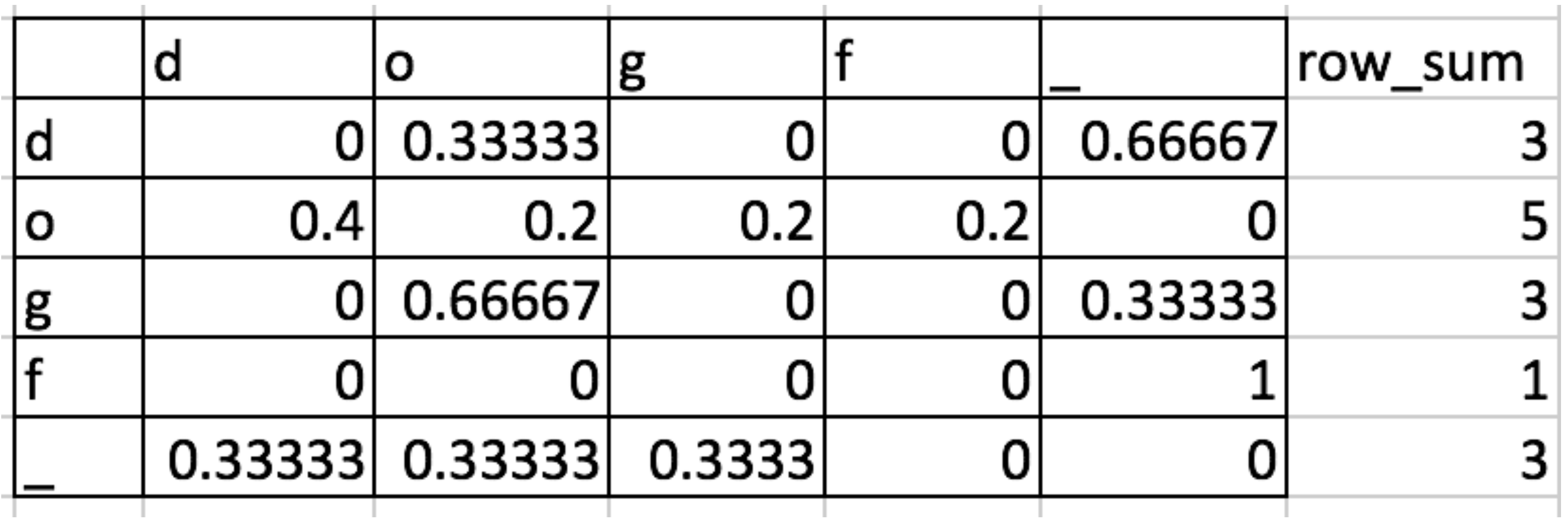
- Quantity of a text making sense in English
- If a text makes, each consecutive 2 letters of the word should correspond to the transition prob matrix

$$\text{Log-like} = \sum_{i=first\ letter}^{last\ letter} \log P(i, i\_next\_letter)$$

- Where i and j correspond to the labels of a row and column of Transition Probability Matrix, P.

Step 2 - Defining Log Likelihood - Example

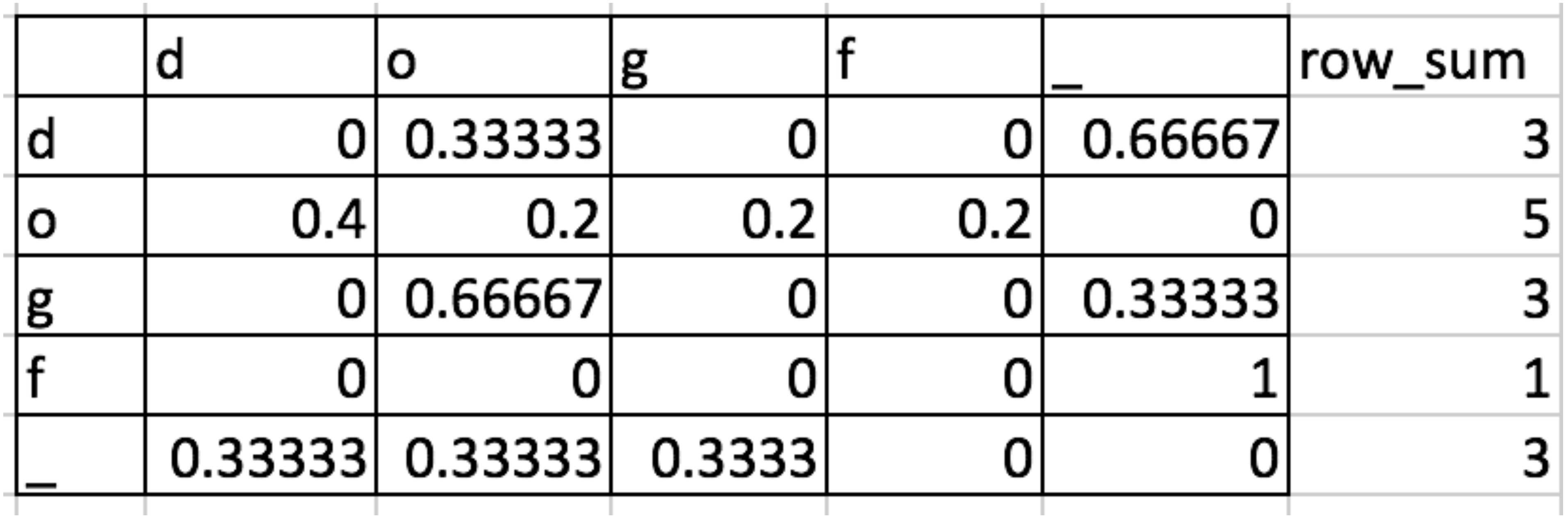
- loglikehood(good) =
  - log(P(g,o)) + log(P(o,o)) + log(P(o,d))
  - log(0.66) + log(0.2) + log(0.4) = **-2.94**
- loglikehood(**og+g**) = ?



	d	o	g	f	_	row_sum
d	0	0.33333	0	0	0.66667	3
o	0.4	0.2	0.2	0.2	0	5
g	0	0.66667	0	0	0.33333	3
f	0	0	0	0	1	1
_	0.33333	0.33333	0.3333	0	0	3

Step 2 - Defining Log Likelihood - Example

- loglikehood(good) =
  - log(P(g,o)) + log(P(o,o)) + log(P(o,d))
  - log(0.66) + log(0.2) + log(0.4) = **-2.94**
- loglikehood(**og+g**) =
  - log(0.2) + log(0.333) + log(0.333)= **-3.81**

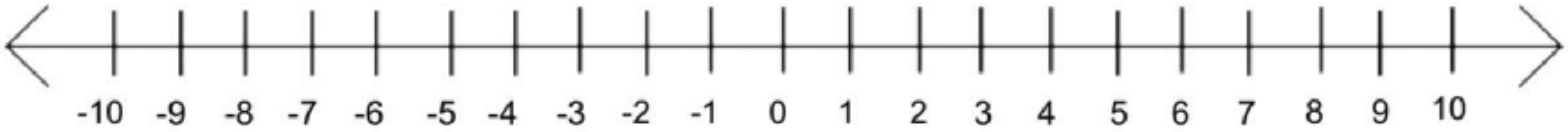


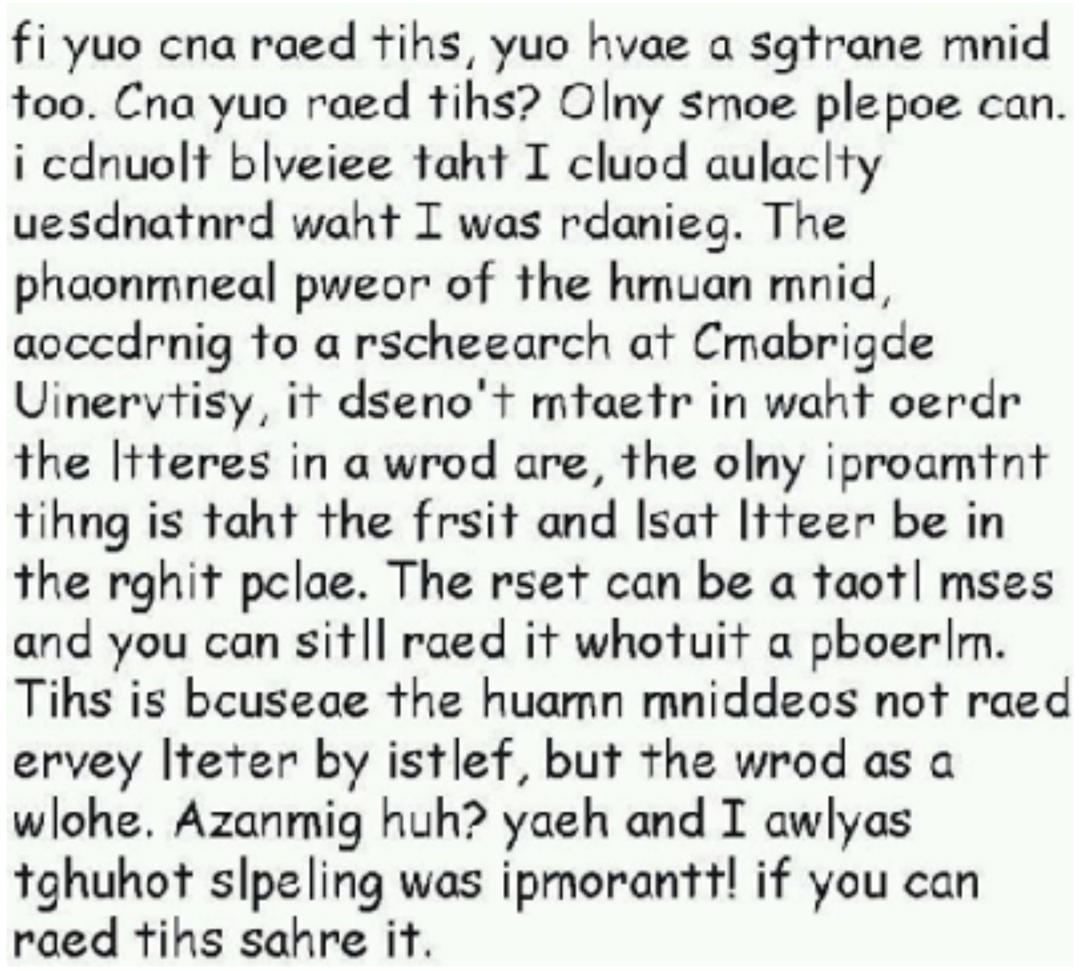
	d	o	g	f	_	row_sum
d	0	0.33333	0	0	0.66667	3
o	0.4	0.2	0.2	0.2	0	5
g	0	0.66667	0	0	0.33333	3
f	0	0	0	0	1	1
_	0.33333	0.33333	0.3333	0	0	3

Step 2 - Defining Log Likelihood

Less sense

More sense





fi yuo cna raed tihs, yuo hvae a sgtrane mnid too. Cna yuo raed tihs? Olny smoe plepoe can. i cdnuolt blveiee taht I cluod aulaclty uesdnatnrd waht I was rdanieg. The phaonmneal pweor of the hmuan mnid, aoccdrnig to a rscheearch at Cmabrigde Uinervtisy, it dseno't mtaetr in waht oerdr the ltteres in a wrod are, the olny iproamtnt tihng is taht the frsit and lsat ltteer be in the rghit pclae. The rset can be a taotl mses and you can sitll raed it whotuit a pboerlm. Tihs is bcuseae the huamn mnidde os not raed ervey lteter by istlef, but the wrod as a wlohe. Azanmig huh? yaeh and I awlyas tghuhot slpeling was ipmorantt! if you can raed tihs sahre it.

Aside: If you ask a Mathematician?

- Treat like Global Optimization Problem
    1. Define Objective Function (Log-likelihood)
    2. Find a mapping (deciphering) to maximize objective fun
    3. **Assumptions:**
        - continuous function,
        - finite decision vectors
        - decision vectors are non-empty

Those assumptions are not always practical....

Step 3 - Metropolis Algorithm - Assumptions

- Proposal Distribution is symmetric.
- In our case, satisfied (explain later).

Step 3 - Metropolis Algorithm - Warm Up

- Probabilistic Model, random number generation
- Example Encoding: 'a' > 'a', 'b' > 'b', 'd' > 'd' ... 'z' > 'z'
- 'bad' > ?

- Scrambled letters 'a' > 'b', 'b' > 'a', 'd' > 'c', ... 'z' > 'z'
- 'bad' > ?
- Mess up letters > New mapping

Step 3 - Metropolis Algorithm

- Simply, using random number generation, start with a random scrambled mapping (**current mapping**)
- Compute log-like of decoded text, current mapping (**current log-like**)
- Randomly swap **Two** letters of current mapping to propose a **new mapping**
- Compute log-like of proposed mapping

Step 3 - Metropolis Algorithm

- Probability to accept is Normalized log-like ratio of the new mapping and the current mapping

```
P(accept proposed) = exp(Log-like(proposed) - Log-like(current))
```

- RNG to decide accept or not

- If accepted, update current mapping with Proposed. **Repeat**

Step 3 - Metropolis Algorithm - Example

- True mapping (encoding): d>o, o>f ,g>d, f>g
- 'dog' >> 'ofd' (encoded text to decipher)
- Initial mapping (random, 4 letters): d>g, o>d, g>f, f>o
- Decoded based on current mapping: ‘fgo'
- log-like(current) = log(0.01) + log(0.667) = **-5.01**
- proposed mapping (g, d): d>d, o>g, g>f, f>o
- Decoded based on proposed mapping: ‘fgd'
- log-like(proposed) = log(0.01) + log(0.01) = **-9.21**

Step 3 - Metropolis Algorithm - Example

- Accept the proposed mapping with the probability

```
P(accept proposed) = exp(Log-like(proposed) - Log-like(current))
```

- P=exp(-9.21 - 5.01)= 0.0000007

- If proposed mapping gets accepted, update the current mapping with the proposed mapping

- Given enough time and bigger text, the current mapping will eventually converge into the **true mapping (true encoding).**

Fun Example

1. Go to: https://github.com/snooravi/meetups
2. Clone/download the directory to your local computer
3. Open jupyter notebook (run ‘jupyter notebook’ in CLI)
4. Navigate to: decryptation_example/ .ipynb

Further Reading:

- Introduction to Cryptology: here
- Decrypting Classical Cipher Text Using Markov Chain Monte Carlo: here
- Text Decryption Using MCMC: here
- Substitution Ciphers: here