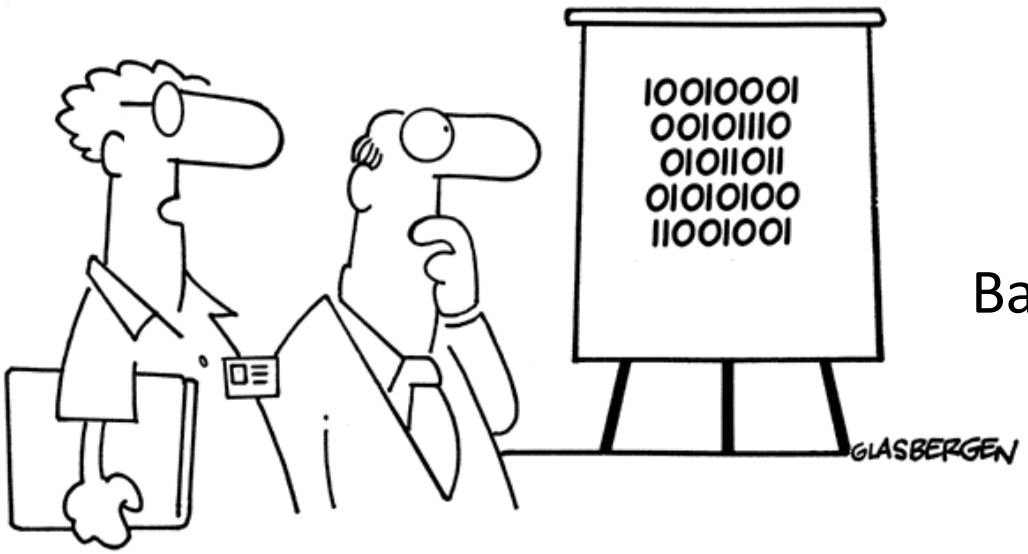
Mini-projet 1: « Cryptographie »



"We've devised a new security encryption code.
Each digit is printed upside down."

Gaultier Lonfat

Barbara Jobstmann

Jamila Sam

Outline

- Administrative
  - Information/Starting point
  - Submission and Groups
  - Submission Server and Tokens
- Project
  - Goal
  - Overview
  - Provided Code
  - Project Details:
    - Part 1: Encryption
    - Part 2: Basic Cryptoanalysis
    - Part 3: Cryptoanalysis based on Frequencies
    - Part Bonus

Information about the Project

- Detailed project description and provided material in Moodle under “Mini-projet 1”

Submission

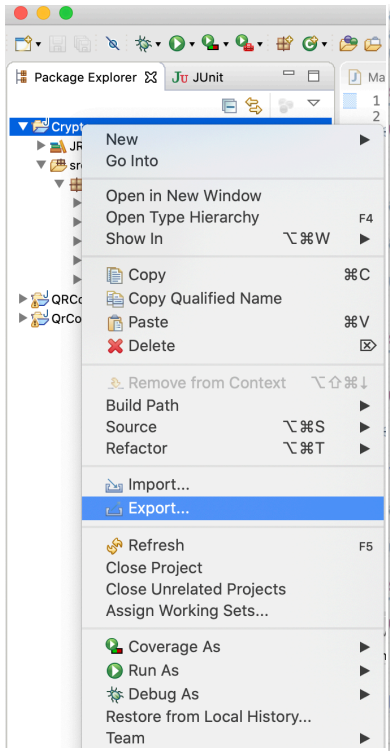
- **Deadline: Tue, Nov 10th, 1pm**
- Groups of (at most) 2 students (not necessarily both in the same A, B or C group)
- Instruction for submission available on Moodle

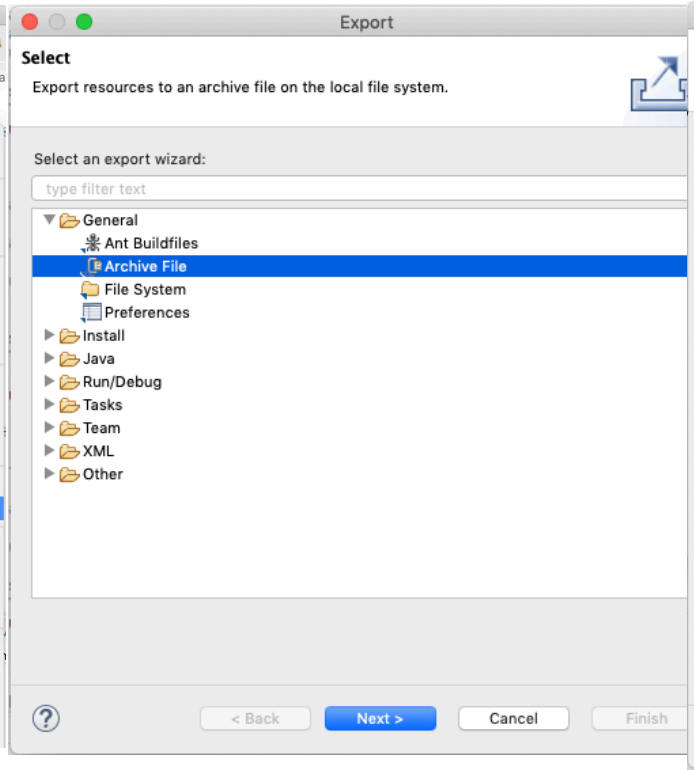
Submission Content

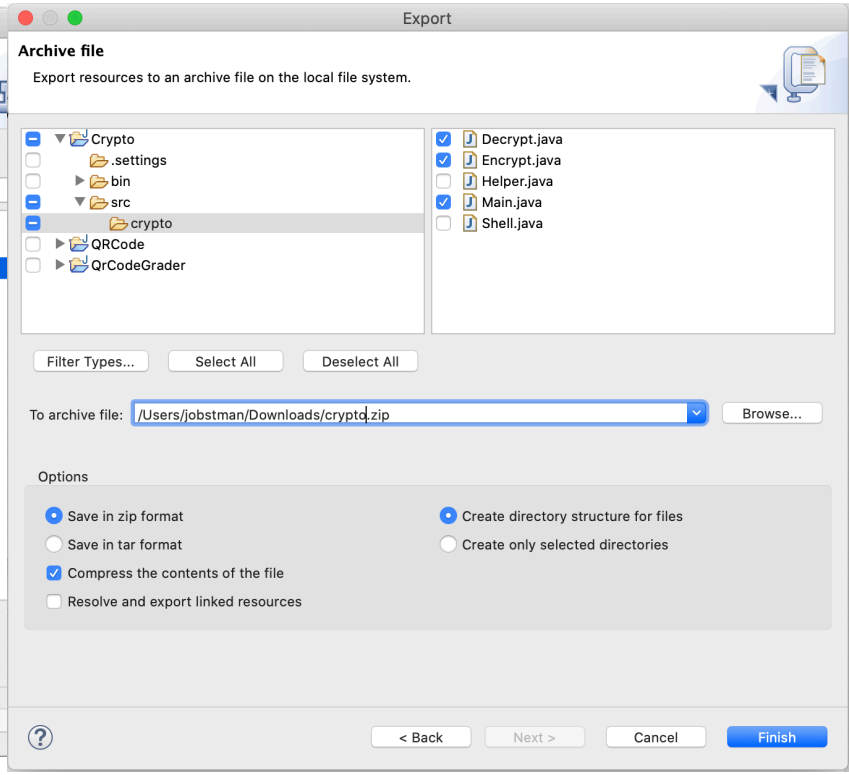
- Eclipse Archive file (zip-file < 20kB) that includes
  - **Encrypt.java (required)**
  - **Decrypt.java (required)**
  - Main.java (optional)
  - README (required if you implement a bonus)
  - Other Java files (for bonus) are optional

Submission Content

- Eclipse Archive file (zip-file < 20kB) that includes
  - **Encrypt.java (required)**
  - **Decrypt.java (required)**
  - Main.java (optional)





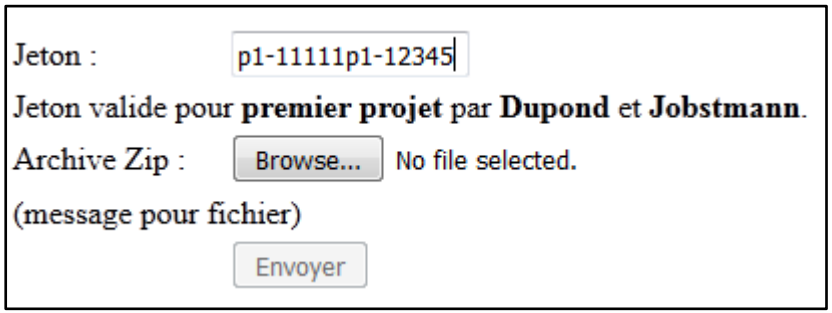


Submission Server

- Will open one week before the deadline:
  - From Tue, Nov 3rd until Fri, Nov 6th 4pm.
  - **No submissions over the weekend!**
  - Reopen on Mon, Nov 9th at 9am and close on Tue, Nov 10th at 1pm (**strict deadline**).
- Each student will need a token (specific key) to submit.
- Tokens will be sent out per email one week before the submission deadline.
- Submission requires two tokens: one from each group member. If you work alone, you need to use your token twice.
- You can submit a new version using the same token.
- **Your 1st Task:** submit initial (incomplete) version **several days before the deadline** to get familiar with the submission process

Submission Server – Examples

- Example tokens: p1-11111 and p1-12345
- Example of submission with 2 students



- Example of submission with 1 student

Submission – Cheating

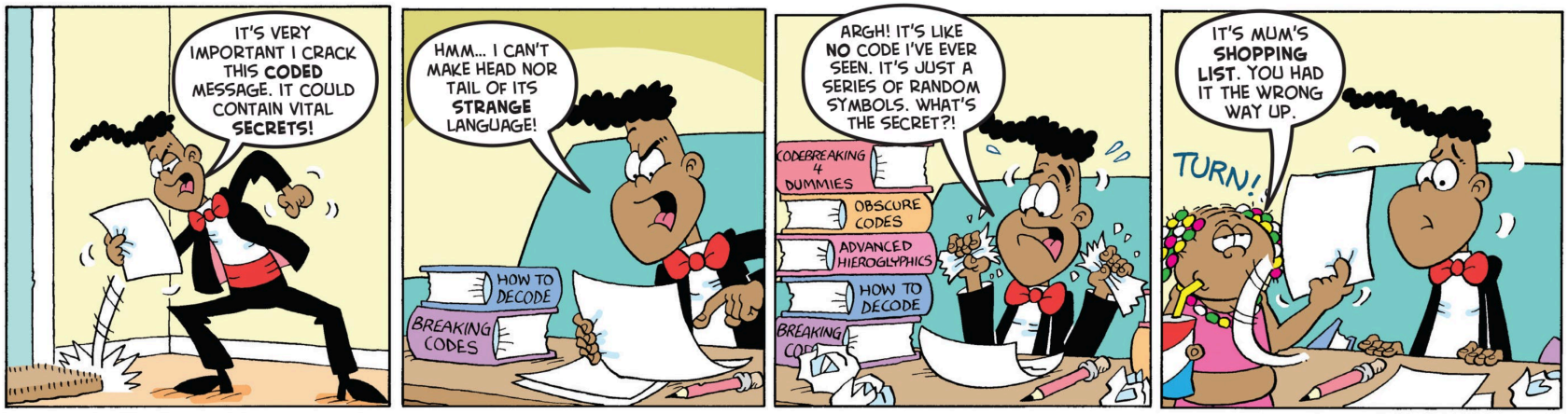
- The project is graded.
- The exchange of ideas between groups or with third parties is permitted and even encouraged.
- **The exchange of code is strictly forbidden!**
- **Plagiarism will be controlled and will be considered cheating.**
- In case of cheating, you will receive a rating of "NA“: Art. 18 “**Fraude de l'ordonnance sur la discipline**“ https://www.admin.ch/opc/fr/classified-compilation/20041650/index.html
- **Note that at any time, you will need to be able to explain your code**.

Outline

- Administrative
  - Information/Starting point
  - Submission and Groups
  - Submission Server and Tokens
- Project
  - Goal
  - Overview
  - Provided Code
  - Project Details:
    - Part 1: Encryption
    - Part 2: Basic Cryptoanalysis
    - Part 3: Cryptoanalysis based on Frequencies
    - Part Bonus

Goal

- Our goal is to introduce you to
  - **Cryptosystems**: methods to make a message incomprehensible to adversaries;
    we focus on the classical cryptosystems
  - **Cryptoanalysis**: methods to break a cryptosystem



Words & Art: Paul Palmer

Some notions:

Plain text: message that will be encrypted
Cipher text: encrypted version of the message
Cipher: an algorithm for performing encryption or decryption
K**ey**: secret information used to encrypt and decrypt

Project Overview 1/2

Part 1: Encryption

- **Example:** plainText = we will meet tomorrow at 11h
  cipherText = zh zloo phhw wrpruurz dw 44k
- **Cryptosystems**: Caesar, Vigenère, XOR, Vernam (OTP), cipher block chaining (CBC)

- Why different cryptosystems?
  - An **ideal cryptosystem** produces a cipherText that looks like a **random sequence** of characters
  - The cryptosystems have different strategies to achieve "better" cipherText:
    - Increase the key size
    - Increase the complexity

Project Overview 2/2

Part 2: Basic Cryptoanalysis

- **Brute-force attacks**
- **Decrypt CBC with key**

Part 3: Cryptoanalysis based on Frequencies

- Break Caesar with frequency analysis
- Break Vigenère with frequency analysis

Part Bonus

Provided Code: Structure

Directory	Content
src/crypto/	**Your code goes here** (see below)
res/	Text files with examples of encrypted and decrypted messages

`class Encrypt` and **`class Decrypt`**

- Headers and empty bodies of the methods that **need** to be implemented. Write your code here!

`class Main`

- Examples of how to call the methods for testing purposes. Extend with your own tests!

`class SignatureChecks`

- Checks that the signatures of the required methods are correct (to simplify automatic testing).
- Does not check any functionality!
- Do not modify!

Provided Code: Functions

`class Helpers`
(Do not modify! Changes will not be taken into account.)

- Read a message from or write a message to a file in the **directory res**

```
String readStringFromFile(String filename)
void writeStringToFile(String text, String filename)
void writeStringToFile(String text, String filename,
boolean append)
```

- Convert String to byte array and vice versa.

```
byte[] stringToBytes(String message)
String bytesToString(byte[] numbers)
```

- Clean String: lower case; replace special characters by space

```
String cleanString(String message)
```

Handling Multiple Files (Classes)

- Up to now all your programs were contained in a single file.
- In this project you will be using **several files**
  - Given a static method m1() defined in a file A.java, and a static method m2() defined in a file B.java,
  - If you want to call m2 in the body of m1 you must use the following syntax; B.m2();
- Example you can write in Main.java:

```
String plainText   = "we will meet tomorrow at 11h";
byte[] plainBytes  = Helper.stringToBytes(plainText);
byte[] cipherBytes = Encrypt.caesar(plainBytes, (byte) 3);
String cipherText  = Helper.bytesToString(cipherBytes);
System.out.println(cipherText);

     // cipherText = “zh zloo phhw wrpruurz dw 44k”
```

Character Encoding

- We will use the **ISO/IEC 8859-1 Latin 1** standard
- Encodes every character with **8-bits** (1 **byte** = 256 values)

2_ 32	SP 32 0020	! 33 0021	" 34 0022	# 35 0023	$ 36 0024	% 37 0025	& 38 0026	' ... 0027	(0028	) 0029	* 002A	+ 002B	, 002C	- 002D	. 002E	/ 002F
3_ 48	0 0030	1 0031	2 0032	3 0033	4 0034	5 0035	6 0036	7 0037	8 0038	9 0039	: 003A	; 003B	< 003C	= 003D	> 003E	? 003F
4_ 64	@ 0040	A 65 0041	B 66 0042	C 67 0043	D 68 0044	E 69 0045	F 70 0046	G 71 0047	H ... 0048	I 0049	J 004A	K 004B	L 004C	M 004D	N 004E	O 004F
5_ 80	P 0050	Q 0051	R 0052	S 0053	T 0054	U 0055	V 0056	W 0057	X 0058	Y 0059	Z 005A	[005B	\ 005C	] 005D	^ 005E	_ 005F
6_ 96	` 0060	a 97 0061	b 98 0062	c 99 0063	d 100 0064	e 101 0065	f 102 0066	g 103 0067	h ... 0068	i 0069	j 006A	k 006B	l 006C	m 006D	n 006E	o 006F
7_ 112	p 0070	q 0071	r 0072	s 0073	t 0074	u 0075	v 0076	w 0077	x 0078	y 0079	z 007A	{ 007B	| 007C	} 007D	~ 007E	

E_ 224	à 224 00E0	á 225 00E1	â 226 00E2	ã 227 00E3	ä 228 00E4	å 229 00E5	æ 230 00E6	ç ... 00E7	è 00E8	é 00E9	ê 00EA	ë 00EB	ì 00EC	í 00ED	î 00EE	ï 00EF

Character Encoding

- We convert characters into bytes (with Helper.stringToBytes) and then we will work with bytes.

```
String plainText   = "we will meet tomorrow at 11h";
byte[] plainBytes  = Helper.stringToBytes(plainText);

// plainBytes = {119, 101, 32, 119, 105, 108, 108, 32, 109, 101,
101, 116, 32, 116, 111, 109, 111, 114, 114, 111, 119, 32, 97, 116,
32, 49, 49, 104}
```

- Bytes in JAVA (described in Appendix)
  - 8 bits signed
  - **Range: -128 to 127 (256 different values)**

```
String plainText   = "bonne journée";
byte[] plainBytes  = Helper.stringToBytes(plainText);

// plainBytes = {98, 111, 110, 110, 101, 32, 106, 111, 117, 114,
110, -23, 101}
// -23 corresponds to 256-23 = 233 = é
```

Refresher: Numbers in Java

- All numbers in JAVA are signed! (**2s complement** for negative numbers)
- Numeric literals are by default int or double (in base 10)
  - `13   //int`
  - `3.5  //double`
  - `0b11 //int in base 2  (binary)`
  - `0xFF //int in base 16 (hexadecimal)`
- Arithmetic (and bitwise) operators on bytes return integers, e.g.,
  - `byte b = 10;`
  - `byte sum = b + b;            //compile error`
  - `byte sum = (byte) (b + b); //cast necessary`
- An integer is represented with 4 bytes (32 bits).
  The cast from int to byte (**byte**) removes the higher 3 bytes.
- The cast from byte to int (**int**) fills the higher 3 bytes with the **sign bit**, i.e., 0 for positive and 1 for negative numbers
  - `byte b = 127;        // 0b01111111`
  - `int  i = 127 + 1;    // 0b00000000..10000000 =  128`
  - `byte b2 = (byte) i ;// 0b~~00000000..~~10000000 = –128`

Outline

- Administrative
  - Information/Starting point
  - Submission and Groups
  - Submission Server and Tokens
- Project
  - Goal
  - Overview
  - Provided Code
  - Project Details:
    - Part 1: Encryption
    - Part 2: Basic Cryptoanalysis
    - Part 3: Cryptoanalysis based on Frequencies
    - Part Bonus

Part 1: Encryption – Caesar Cipher

- Caesar Cipher:
  - Shift characters (given as bytes) by a given value (key = 1 byte)
  - Wrap around: 127 + 1 = -128
  - Space (32) is only shifted if third parameter is true

Encrypt space? true/false

```
String plainText   = "bonne journée";
byte[] plainBytes  = Helper.stringToBytes(plainText);
//98, 111, 110, 110, 101, 32, 106, 111, 117, 114, 110, -23, 101

byte[] cipherBytes = Encrypt.caesar(plainBytes, (byte) 3, false);

//101, 114, 113, 113, 104, 32, 109, 114, 120, 117, 113, -20, 104
String cipherText = Helper.bytesToString(cipherBytes);

// if space is not shifted: cipherText = erqqh mrxuqìh
// if space is shifted:     cipherText = erqqh#mrxuqìh
```

In order to decrypt **encrypt with the reverse key**, e.g., encrypt key 3 => decrypt key -3
Note we could also shift by 253 to reverse the effect of 3, since we have 256 characters!

Part 1: Encryption – Vigenère Cipher

- Vigenère Cipher:
  - Shift characters (bytes) by different values (key = byte array)
  - Wrap around
  - Space (32) is only shifted if third input is true

Encrypt space? true/false

```
String plainText   = "bonne journée";
byte[] plainBytes  = Helper.stringToBytes(plainText);
//98, 111, 110, 110, 101, 32, 106, 111, 117, 114, 110, -23, 101
byte[] key = {(byte) 1, (byte) 2, (byte) 3 };

byte[] cipherBytes = Encrypt.vigenere(plainBytes, key, false);

//99, 113, 113, 111, 103, 32, 109, 112, 119, 117, 111, -21, 104

String cipherText = Helper.bytesToString(cipherBytes);
// if space is not shifted: cipherText = cqqog mpwuoëh
// if space is shifted:     cipherText = cqqog#kqxspìf
```

To decrypt **encrypt with reverse key**!

Part 1: Encryption – XOR and OTP

- **XOR Cipher**:
  - XOR – exclusive or: a bit-wise operator, symbol in JAVA ^
    `x = 0b1010 ^ 0b0011; //x = 0b1001 (int!)`
  - XOR each characters (bytes) with a key (byte)
  - Shift of space (32) is optional

Encrypt space? true/false

```
byte[] plainBytes  =
{98, 111, 110, 110, 101, 32, 106, 111, 117, 114, 110, -23, 101};

byte[] cipherBytes = Encrypt.xor(plainBytes, (byte) 7, false);

//flip last three bits because 7 = 0b111
//101, 104, 105, 105, 98, 32, 109, 104, 114, 117, 105, -18, 98,
```

- **One-time-pad**: xor with different values (key = byte array = **pad**)

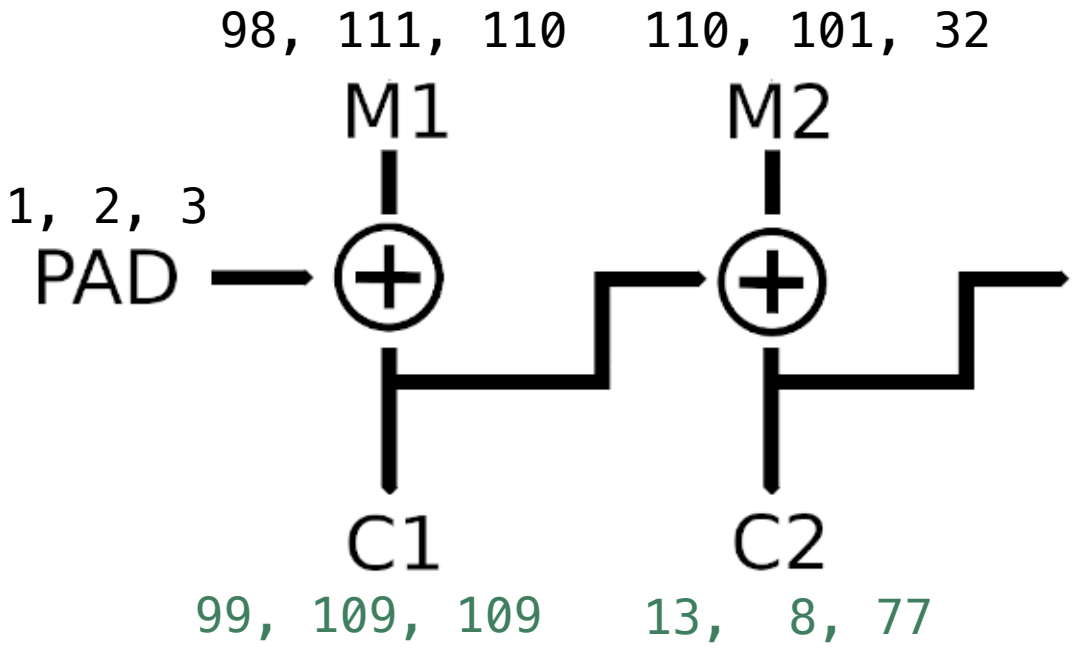
```
byte[] plainBytes  = {98, 111, 110, 110, 101, 32};
byte[] pad         = { 1,   2,   3,   4,   5,  6};

byte[] cipherBytes = Encrypt.oneTimePad(plainBytes, pad);

//cipherBytes = 99, 109, 109, 106, 96, 38
```

To decrypt **encrypt with the same pad** because if you xor twice you get the input back.

Part 1: Encryption – CBC

- **Cipher Block Chaining (simplified version)**:
  - decompose message into blocks of the pad size
  - xor blocks with pad as shown below

```
byte[] plainBytes = {98, 111, 110, 110, 101, 32};
byte[] pad        = { 1,   2,   3};
byte[] cipherBytes = Encrypt.cbc(plainBytes, pad);
                    //99, 109, 109, 13, 8, 77,
```



Part 2: Basic Cryptoanalysis

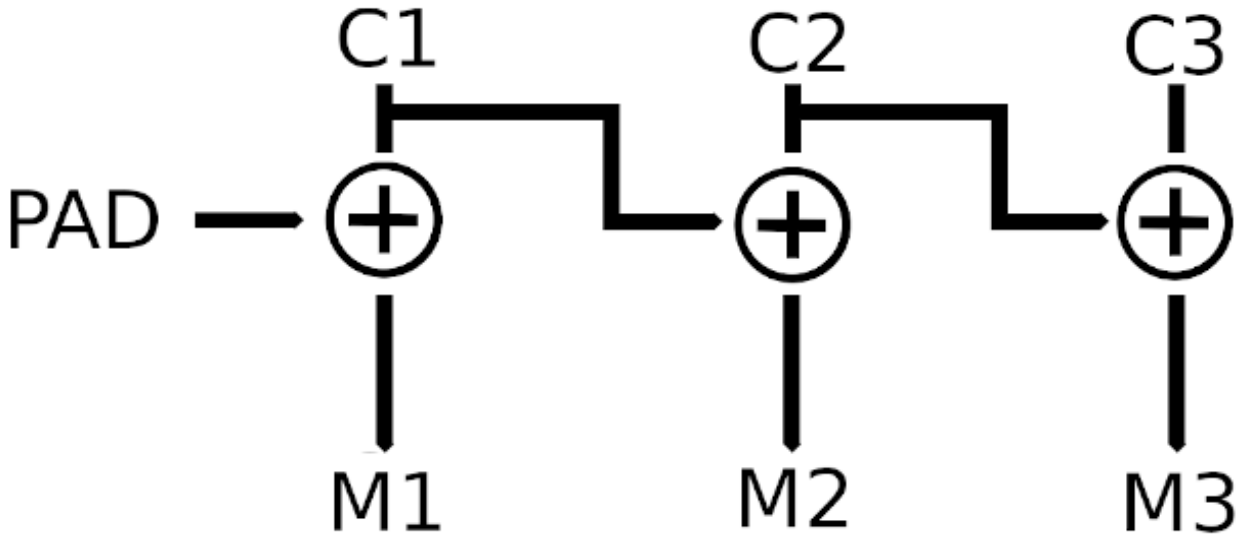
- Break Caesar and XOR with **brute force**
  - Try all possible 256 keys
  - Output: 2-dim byte array with 256 entries, each entry stores the decryption for one key, e.g.,

```
byte[]  cipherBytes = {101, 104, 105, 105, 98};
byte[][] plainBytes =
                    Decrypt.caesarBruteForce(cipherBytes);

plainBytes = { …,
                   {102, 105, 106, 106, 99}, //key = 1
                   {103, 106, 107, 107, 100},//key = 2
                   {104, 107, 108, 108, 101},//key = 3
                   {105, 108, 109, 109, 102},//key = 4
                   …};
```

- Convert byte[][] to String
  - Analyse manually, which of the decoded messages makes sense

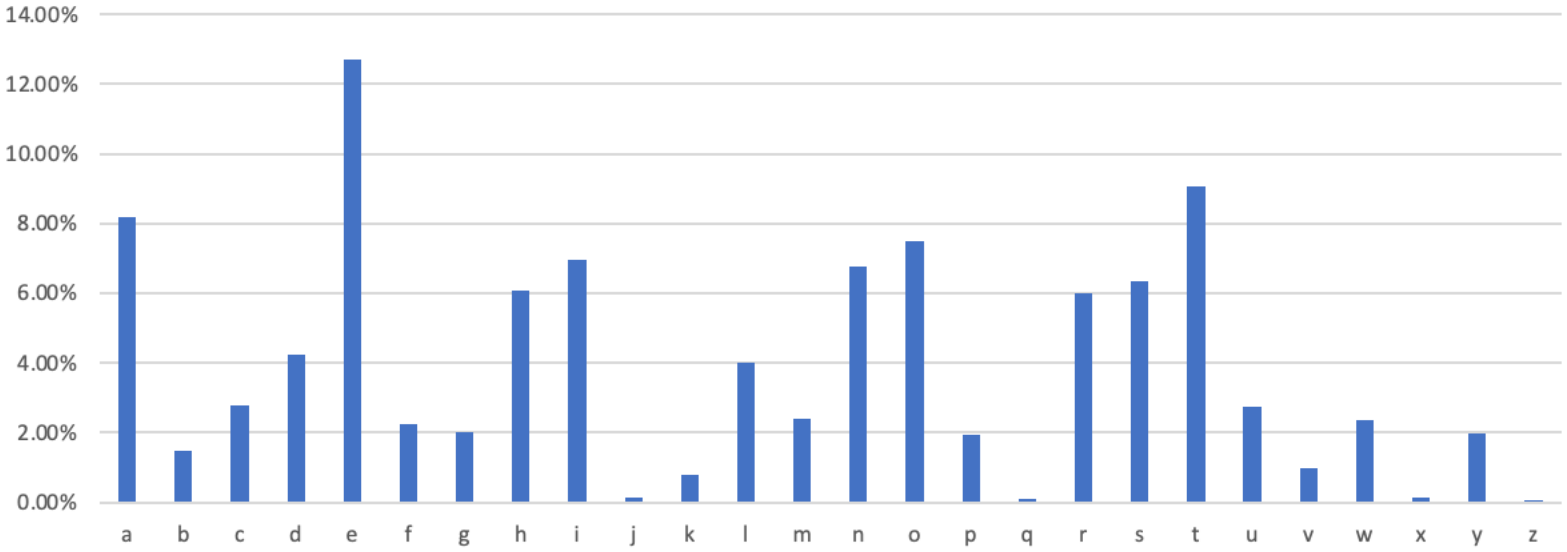
Part 2: Decrypt CBC

- Given a message encrypted using CBC and the corresponding key, compute the decrypted message
  - Decompose the message into blocks of the pad size
  - xor blocks with pad as shown below



Part 3: Cryptoanalysis with Frequencies

- **Key idea**: use the fact that if the message that was encrypted is a sentence in some language, then not all letters are equally likely.
- **Frequency profile** of the English language:

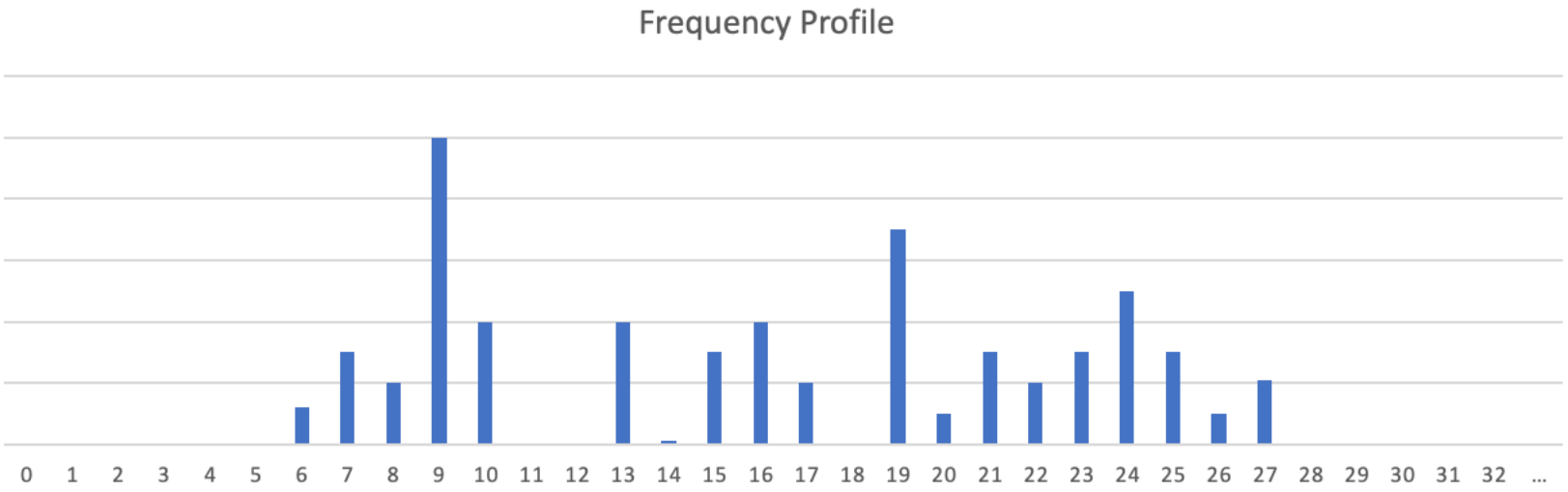


https://en.wikipedia.org/wiki/Letter_frequency

```
double[] ENGLISHFREQUENCIES = {0.08497,0.01492,...};
```

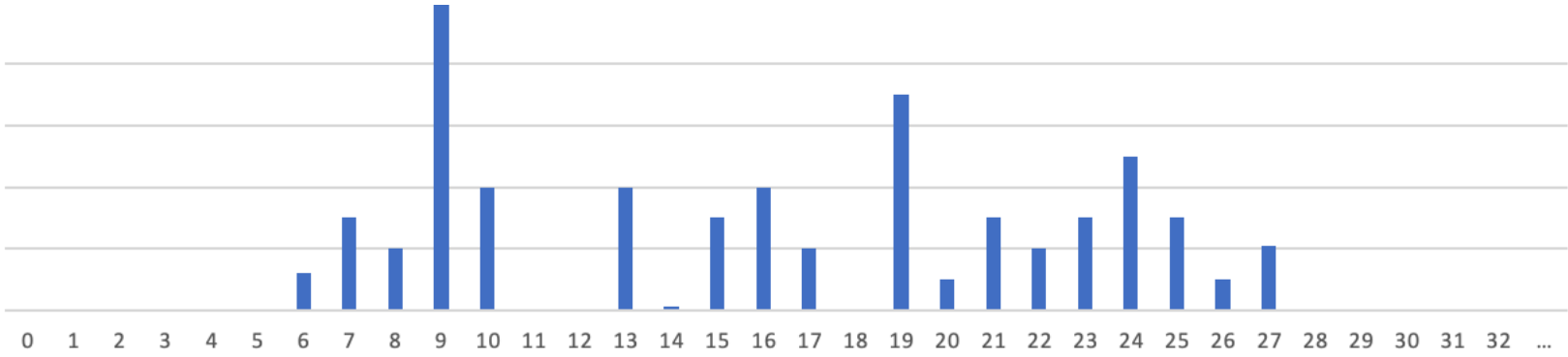
Part 3: Break Caesar (1/2)

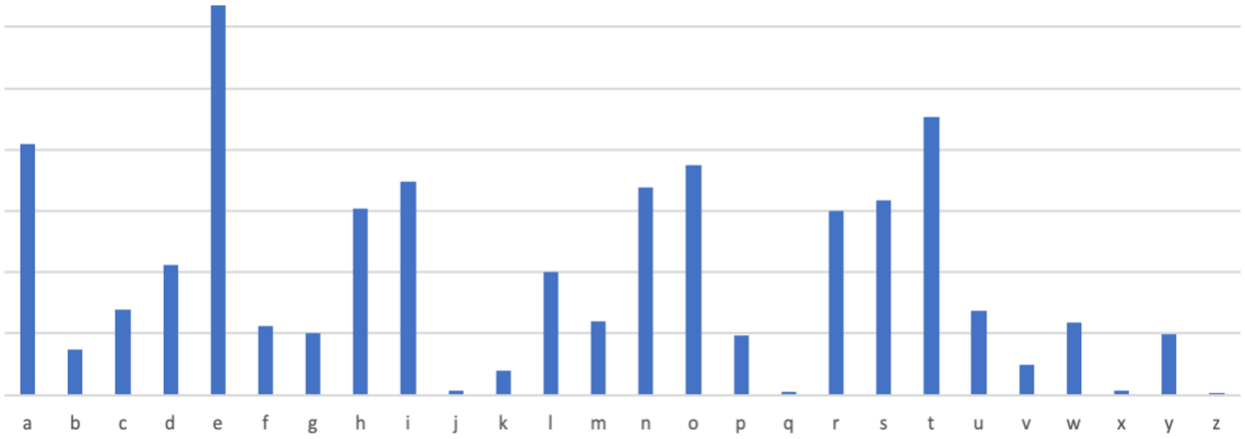
- First, compute the frequency profile of cipherText
  `float[] computeFrequencies(byte[] cipherText)`
  - For each of the 256 characters compute the ratio between the occurrences of this character in the cipher text and the length of the text (ignoring spaces)



Part 3: Break Caesar (2/2)

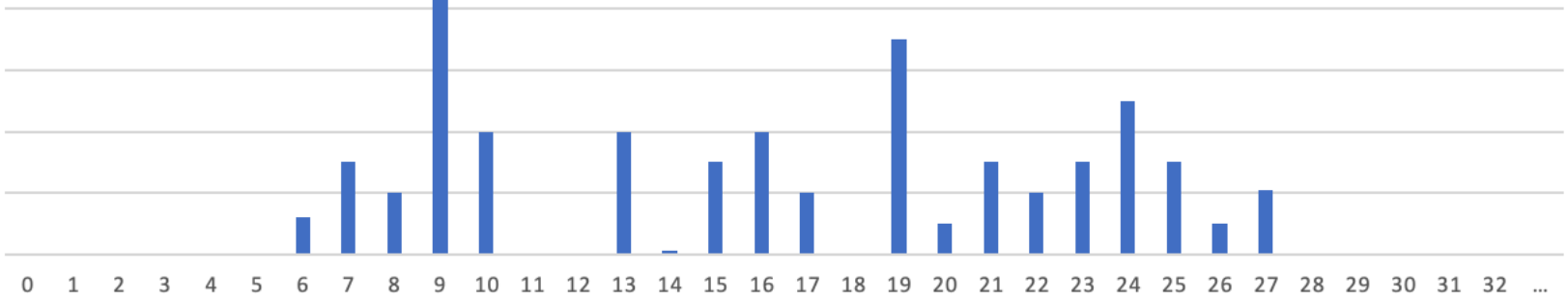
- Find frequency profile alignment
  `byte caesarFindKey(float[] charFrequencies)`
  - By computing the scalar product of given frequencies and the English frequencies for all positions

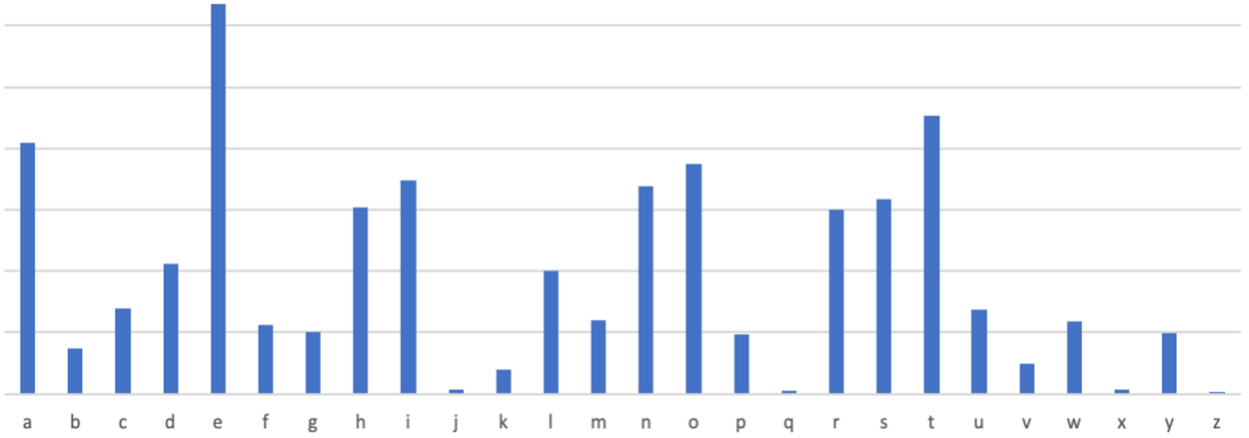




Part 3: Break Caesar (2/2)

- Key is the positive distance (if “a” was shifted to the right) or negative distance (if “a” was shifted to the left) between the index of the optimal alignment (5 below) and the position of letter “a” (which is 97), e.g., 5 - 97 = -92 (“a” was shifted 92 positions to the left)





Part 3: Break Vigenère

- **Idea**: find the key length and then apply break caesar for every entry in the key on a subset of elements
- For instance, assume we know that the key length is 3 and the cipher text looks as follows:

99	113	113	111	103	32	109	112	119	117	111	-21	104

- Then we run 3x Break Caesar:
  - Once with the orange, once with yellow and once with the green elements (spaces are ignored)

99	113	113	111	103	32	109	112	119	117	111	-21	104

99	113	113	111	103	32	109	112	119	117	111	-21	104

99	113	113	111	103	32	109	112	119	117	111	-21	104

Part 3: Find Vigenère Key Length (1/3)

1. Find coincidences
   - Compare the cipher text with a shifted version of itself
   - For each shift compute how many letters coincide (“coincidence”).
   - For instance with a shift of 1, we have 1 coincidence

99	113	113	111	103	32	109	112	119	117	111	-21	104	
	99	113	113	111	103	32	109	112	119	117	111	-21	104

   - With a shift of 2, we have 0 coincidence

99	113	113	111	103	32	109	112	119	117	111	-21	104	
	99	113	113	111	103	32	109	112	119	117	111	-21	

Part 3: Find Vigenère Key Length (2/3)

2. Given the coincidences table, find the **local maxima** in the first half of the table
   - A value is a local maximum if its two neighbours on the right and its two neighbours on the left are smaller

Shift	1	2	3	4	5	6	7	8	9	10	…
Coinc.	1	0	2	3	0	2	3	1	4	5	1

   - Indices of local maxima: 4, 7, 10

Part 3: Find Vigenère Key Length (3/3)

3. Given the indices of local maxima, compute the distance between any two consecutive indices. The key length is the distance that appears the most often.
   - For instance, given the following indices of the local maxima:

Indices	4	7	10	12	15	18	23	26	28	30	…

   - The distances are

Distances	3	3	2	3	3	5	3	2	2	…

   - Distance 3 appears the most often and is therefore (most likely) the **key length**

Part 3: Find Vigenère Key Length When and why does it work?

- When?
  If the input text has a frequency profile close to the reference profile (to the English language in our case), i.e., the text needs to be long enough.
- Why?
  - If you pick two random letters from an English text you have a 6.6% chance to pick the same letter.
  - This chance does not change if you use a Caesar cipher because the frequency profile does not change.
  - For a text with random characters you have only a chance of 3.84% to pick the same letter.
  - Therefore, if a cipherText was encrypted with a key of length m and you compare it with a version that is shifted by a multiple of the key length then you should get a higher number of coincidences than for other shifts.
  - Since we are working with probabilities it makes sense to find all the positions with a high number of coincidences.

Final remarks

- Don’t be scared by the length of the handout!
  - It does not only list the tasks that you have to complete
  - But also some theory and explanations
    - How to do X?
    - Why do it this way?
- Use the **Piazza forum** for questions and discussions
  - DO NOT POST CODE
- TAs will help you if you are stuck or if something is not clear

Bon travail!