Code Breaking

Task 1

Visualization of the dato

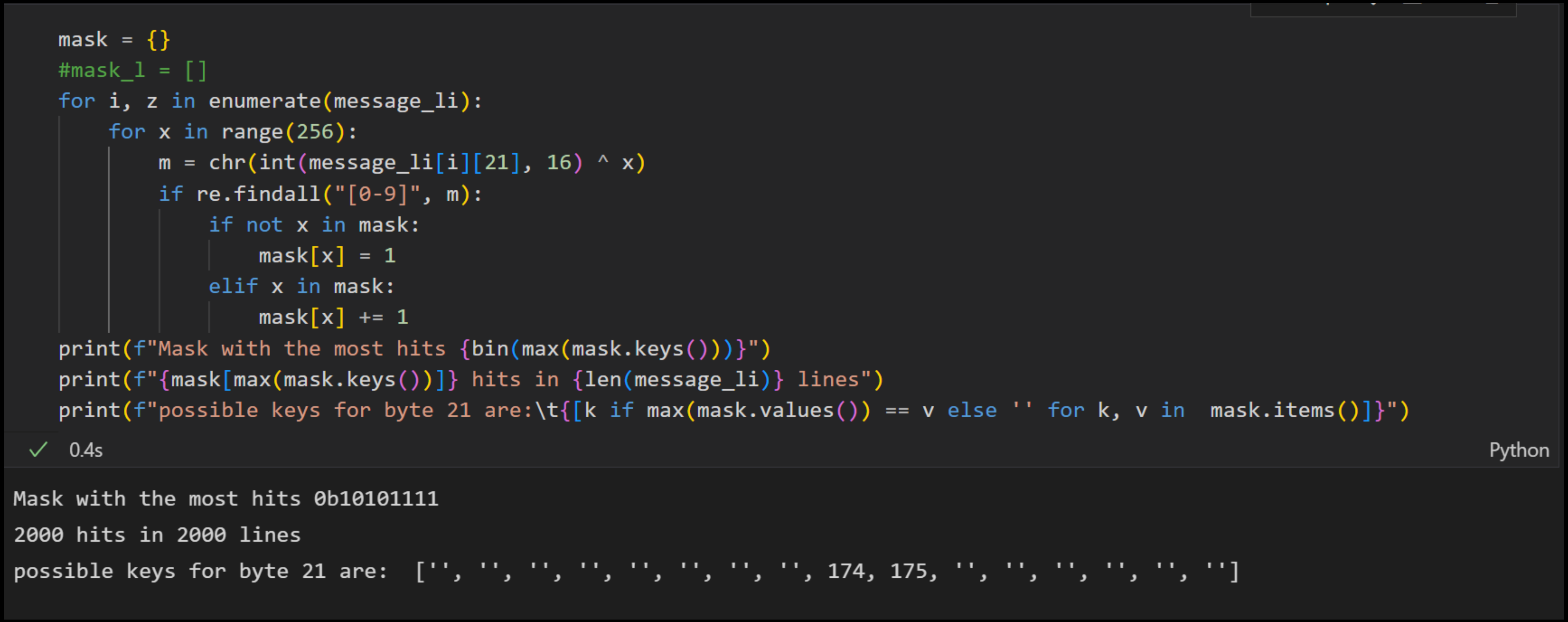
- There is a repeating pattern in the recorded date

```
9d ca 92 f2 b3 22 2a ed 58 85 cd cb 43 15 a1 64 81 e7 54 2d 16 da 2a 66

8d c7 99 e2 b7 39 64 bb 42 d6 db cd 55 12 ba 75

8d c7 99 e2 b7 39 64 b8 42 d6 de cd 44 1e e8 63 9d a4 52 2b 12

8d c7 99 e2 b7 39 64 b9 42 d6 c1 c6 55 15 ad 74 91 a5 4b 20 57 dd 3a 76 2b 91 af fe

8d db 84 e5 bd 26 64 f9 1d 95 dd da 53 47 xx xx xx xx xx xx xx xx

f0 8c d9 b1 af 2a 2d fe 11 98 cf 88 50 08 ba 30 96 a2 5f 2d 57 da 2d 7c 39 84 af ad

9d ca 92 f2 b3 22 2a ed 58 85 cd cb 43 15 a1 64 81 e7 54 2d 16 da 2a 66
```

Visualization of the dato

- As you can see, the 8 last Bytes of the line starting with „8d db“ are varying
- There are 10 different Bytes for the last Byte in the lines starting with „8d db“
    - Possible Numbers from 0 to 9???
- 6 different Bytes for secund last Byte
    - Numbers form 0 to 6???
    - Last two Bytes could be time annotation
    - Same for the fourth and fifth last Byte
- Third last Byte is constant
    - The Text is American/English, time separator is :

Decoding of data



```python
mask = {}
#mask_l = []
for i, z in enumerate(message_li):
    for x in range(256):
        m = chr(int(message_li[i][21], 16) ^ x)
        if re.findall("[0-9]", m):
            if not x in mask:
                mask[x] = 1
            elif x in mask:
                mask[x] += 1
print(f"Mask with the most hits {bin(max(mask.keys()))}")
print(f"{mask[max(mask.keys())]} hits in {len(message_li)} lines")
print(f"possible keys for byte 21 are:\t{[k if max(mask.values()) == v else '' for k, v in  mask.items()]}")
```

✓ 0.4s Python

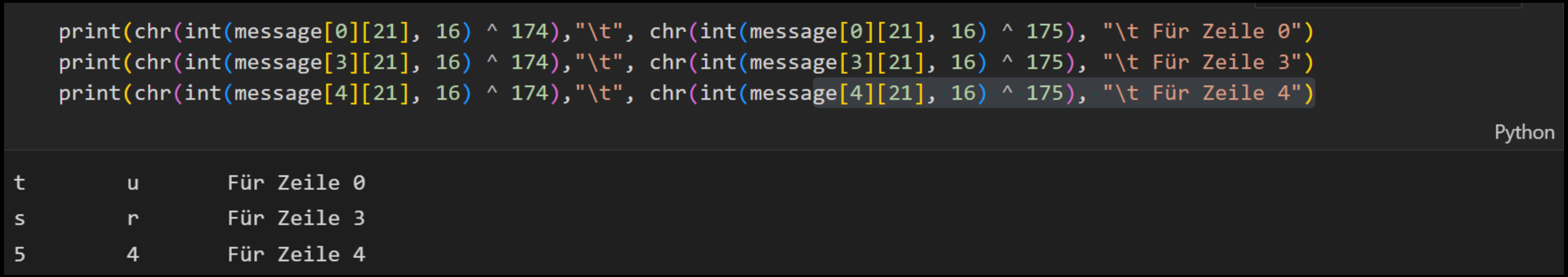
```
Mask with the most hits 0b10101111
2000 hits in 2000 lines
possible keys for byte 21 are:  ['', '', '', '', '', '', '', '', 174, 175, '', '', '', '', '', '']
```

Decoding of data

- Mask with the most hits 0b10101111 2000 hits in 2000 lines possible keys for byte 21 are:
- ['', '', '', '', '', '', '', '', 174, 175, '', '', '', '', '', ‘’]

- We got two possible hits

Decoding of data

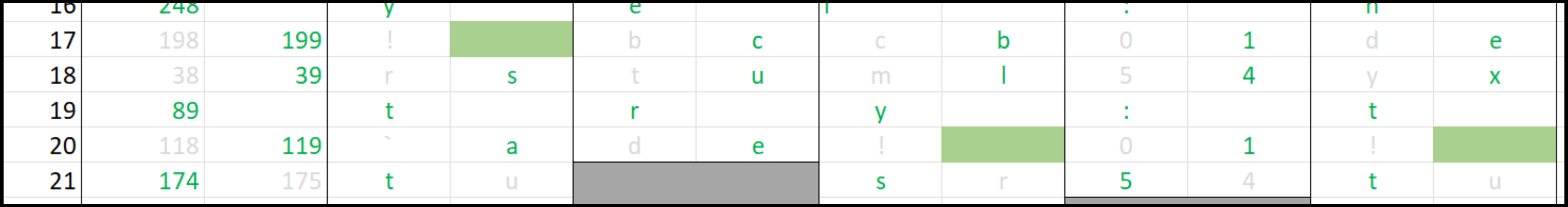


```python
print(chr(int(message[0][21], 16) ^ 174),"\t", chr(int(message[0][21], 16) ^ 175), "\t Für Zeile 0")
print(chr(int(message[3][21], 16) ^ 174),"\t", chr(int(message[3][21], 16) ^ 175), "\t Für Zeile 3")
print(chr(int(message[4][21], 16) ^ 174),"\t", chr(int(message[4][21], 16) ^ 175), "\t Für Zeile 4")
```

```
t        u        Für Zeile 0
s        r        Für Zeile 3
5        4        Für Zeile 4
```

Decoding of data

- Adding possible Keys and decoded dato to an Excel spreadsheet



16	248		y		e		l		:		n	
17	198	199	!	🟩	b	c	c	b	0	1	d	e
18	38	39	r	s	t	u	m	l	5	4	y	x
19	89		t		r		y		:		t	
20	118	119	`	a	d	e	!	🟩	0	1	!	🟩
21	174	175	t	u	▨	▨	s	r	5	4	t	u

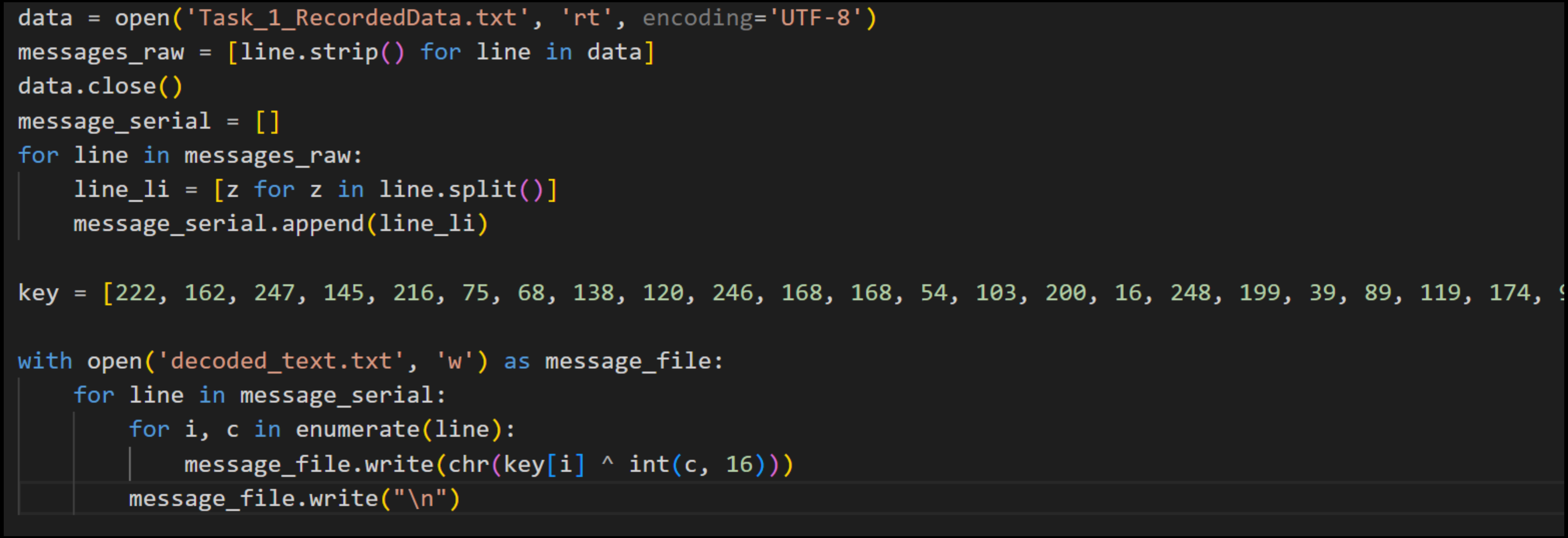
- Decoding each line wit the given Keys
- Words can now be guessed in order to gdecide which key is being used
- From the word „cure“ can bthe word „secure“ be guessed
- A new key can be brutforced from the new letter

Decoding of data

Bytes	Maske		Mögliche Zeichen									
		Zeile →	1		3		4		5		6	
0	222		C		S		S		S		.	
1	162		h		e		e		y		.	
2	247		e		n		n		s		.	
3	145		c		s		s		t			
4	216		k		o		o		e		w	
5	75		i		r		r		m		a	
6	68		n								i	
7	138		g		2		3		s		t	
8	120				:		:		e		i	
9	246		s						c		n	
10	168		e		v		i		u		g	
11	168		c		e		n		r			
12	54		u		r		c		e		f	
13	103		r		y		r				o	
14	200		i				e		o		r	
15	16		t		s		d		3			
16	248		y		e		i		:		n	
17	198	199	!		b	c	c	b	o	1	d	e
18	38	39	r	s	t	u	m	l	5	4	y	x
19	89		t		r		y		:		t	
20	118	119	`	a	d	e	!		o	1	!	
21	174	175	t	u			s	r	5	4	t	u
22	95		u				e				r	
23	21		s				c				i	
24	94						u				g	
25	227						r				g	
26	202						e				e	
27	223						!				r	

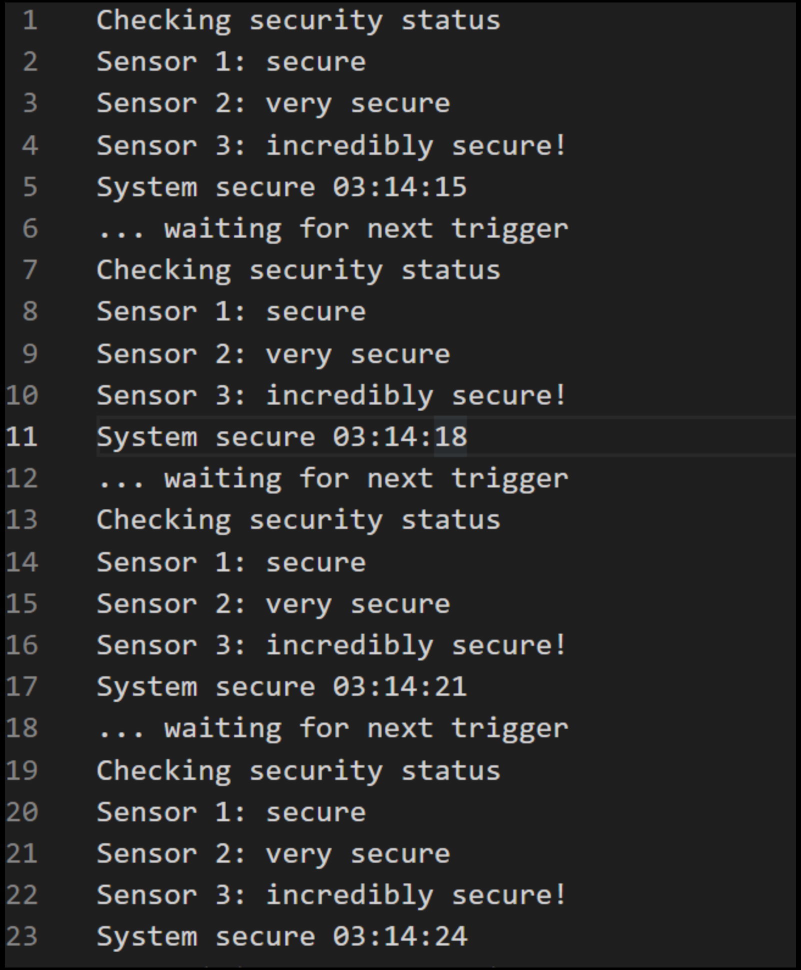
Decoding of data

With the key form the Excel spreadsheet, we can decode the whole recorded data



```python
data = open('Task_1_RecordedData.txt', 'rt', encoding='UTF-8')
messages_raw = [line.strip() for line in data]
data.close()
message_serial = []
for line in messages_raw:
    line_li = [z for z in line.split()]
    message_serial.append(line_li)

key = [222, 162, 247, 145, 216, 75, 68, 138, 120, 246, 168, 168, 54, 103, 200, 16, 248, 199, 39, 89, 119, 174,

with open('decoded_text.txt', 'w') as message_file:
    for line in message_serial:
        for i, c in enumerate(line):
            message_file.write(chr(key[i] ^ int(c, 16)))
        message_file.write("\n")
```

Decoded data



```
Checking security status
Sensor 1: secure
Sensor 2: very secure
Sensor 3: incredibly secure!
System secure 03:14:15
... waiting for next trigger
Checking security status
Sensor 1: secure
Sensor 2: very secure
Sensor 3: incredibly secure!
System secure 03:14:18
... waiting for next trigger
Checking security status
Sensor 1: secure
Sensor 2: very secure
Sensor 3: incredibly secure!
System secure 03:14:21
... waiting for next trigger
Checking security status
Sensor 1: secure
Sensor 2: very secure
Sensor 3: incredibly secure!
System secure 03:14:24
```

Exploited weaknesses

1. Enough recorded data was given
2. We knew some of the meaning from the ciphertext
3. Rest of the word could be guessed after partial decoding
4. Same key was used for the same byte in every line

Promising counter measures

1. Use random padding
2. Use different keys for every byte. Even for different lines
3. Use a block cipher
4. Use multiple keys to make brut forcing more difficult