Features Business Explore Pricing This repository Search Sign in or Sign up

USCGA / writeups Watch 3 Star 6 Fork 2

Code Issues 0 Pull requests 0 Projects 0 Pulse Graphs

Branch: master writeups / online_ctfs / qiwi_infosec_ctf_2016 / crypto_100_3_COMPLETE / Create new file Find file History

JohnHammond readded everything Latest commit 9a609f9 on 6 Feb

..

README.md	readded everything	2 months ago
dna_chart.jpg	readded everything	2 months ago
dna_solver.py	readded everything	2 months ago

README.md

Crypto 100_3

> John Hammond | Friday, November 18th, 2016

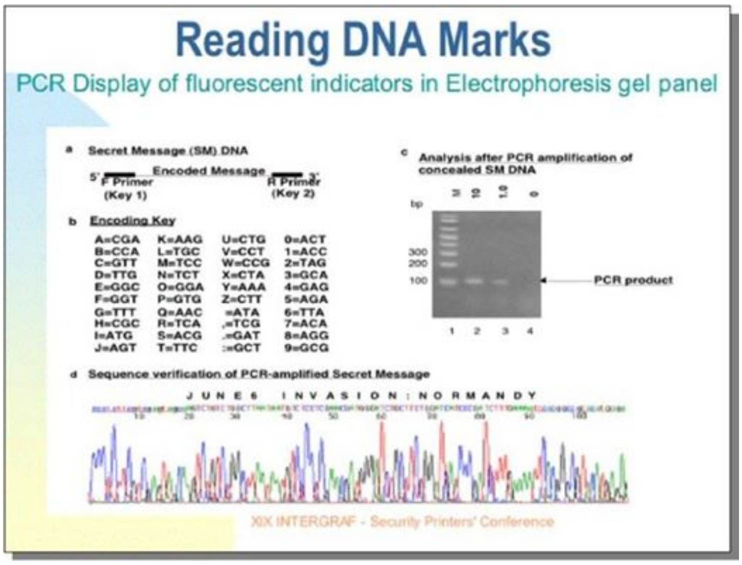
> The flag is a plaintext.
>
> **Ciphertext:** `GGTTCAATGGGCTTGTCAATGGTTCGCATATCCATGGGCACGGTTCGCGGCTCA`
>
> **Hint1:** Change space to `_`

So we're given no background or description... just a ciphertext and a notion that the flag is plaintext.

Well great. What do we do? This ciphertext clearly isn't some kind of easily caesar cipher or vignere cipher or anything like that. Do the characters represent other characters, like a substitution cipher? Well if you count, there are only *four* letters, so there is no way that could work...

But.... after staring at it for long enough... notice that those *four* letters in use are `G`, `T`, `C`. and `A`... the letters used for DNA!

After some hunting on the Internet, this chart surfaced.



Reading DNA Marks

PCR Display of fluorescent indicators in Electrophoresis gel panel

a Secret Message (SM) DNA

b Encoding Key

A=CGA	K=AAG	U=CTG	0=ACT
B=CCA	L=TGC	V=CCT	1=ACC
C=GTT	M=TCC	W=CCG	2=TAG
D=TTG	N=TCT	X=CTA	3=GCA
E=GGC	O=GGA	Y=AAA	4=GAG
F=GGT	P=GTG	Z=CTT	5=AGA
G=TTT	Q=AAC	=ATA	6=TTA
H=CGC	R=TCA	,=TCG	7=ACA
I=ATG	S=ACG	.=GAT	8=AGG
J=AGT	T=TTC	:=GCT	9=GCG

c Analysis after PCR amplification of concealed SM DNA

d Sequence verification of PCR-amplified Secret Message

XIX INTERGRAF - Security Printers' Conference

It gave us a mapping, and I went ahead and turned this into a Python dictionary.

```
mapping = {
        'CGA': 'A',
        'CCA': 'B',
        'GTT': 'C',
        'TTG': 'D',
        'GGC': 'E',
        'GGT': 'F',
        'TTT': 'G',
        'CGC': 'H',
        'ATG': 'I',
        'AGT': 'J',
        'AAG': 'K',
        'TGC': 'L',
        'TCC': 'M',
        'TCT': 'N',
        'GGA': 'O',
        'GTG': 'P',
        'AAC': 'Q',
        'TCA': 'R',
        'ACG': 'S',
        'TTC': 'T',
        'CTG': 'U',
        'CCT': 'V',
        'CCG': 'W',
        'CTA': 'X',
        'AAA': 'Y',
        'CTT': 'Z',
        'ATA': ' ',
        'TCG': ',',
        'GAT': '.',
        'GCT': ':',
        'ACT': '0',
        'ACC': '1',
        'TAG': '2',
        'GCA': '3',
        'GAG': '4',
        'AGA': '5',
        'TTA': '6',
        'ACA': '7',
        'AGG': '8',
        'GCG': '9'
}
```

Easy enough. Now, all we had to do was map each section...

```
def decode_dna( string ):

    pieces = []
    for i in range( 0, len(string), 3 ):
        piece =  string[i:i+3]
        # pieces.append()
        pieces.append( mapping[piece] )

    return "".join(pieces)


string = 'GGTTCAATGGGCTTGTCAATGGTTCGCATATCCATGGGCACGGTTCGCGGCTCA'
print decode_dna(string)
```

This yields the flag: ... but ***we have to remember what the hint says: change the space to an underscore!***

```
FRIEDRICH_MIESCHER
```