DeapSECURE Crypto Challenge 1

DeapSECURE workshop series, 2020-2021

Cracking AES Ciphertexts (due: Fri 11/20 @ 12:00pm)

Imagine that you are working for a secret agent named Jack Snooper who went to Norfolk to explore ODU without being caught. Jack is sending streams of encrypted messages back to your company to tell you where he is, what he is seeing, etc. He does not want to prearrange the secret keys before embarking his mission. He only told you that each message will be 128-bit long, encrypted with **AES-128**, and he will tell you the number of unknown bits the key will have. Other higher bits will be zeros. So, you have to perform AES cracking for every secret message. You have an HPC to perform this duty, but you must decrypt the messages as fast as you can, because they are critical to your mission. Jack sent a lot of messages with varying difficulty levels, and you must break all the messages. The keys have either 8, 16, 20, 24, and 32 unknown bits.

Your challenge is to recover as many original messages as possible in the shortest amount of time. For each message below, you can be a winner in one or more categories:

1. **Earliest Decipher**: You are the "Earliest Decipher" winner if you are the first to submit the correct answer for a given ciphertext.
2. **Fastest Decipher**: You are the "Fastest Decipher" winner if your program obtains the correct answer in the shortest amount of running time.
3. **Most Creative Solution**: You will get this award if you deploy interesting or unusual ways to solve the ciphertexts.

We want you to turn in the original message, the key used to crack that message, and (optionally) how long your program takes to recover the message. Please carefully read the complete rules of the competition below (this page and next two pages) before you begin your effort.

Here is an example of a secret message (ciphertext):

Key bits	Ciphertext	Plaintext
8	0xd3bbd04550497f943f6bf4c9e2291993	CongratsYouDidIt

The cracking process should reveal the AES key used to crack that message and the resulting plaintext message. In this example, you need to crack the lowest 8 bits of the key. The higher 120 bits of the key are zeros. Note that each message in the challenge has its own secret key.

The Valid Messages

Valid messages will have the following characteristics:

- Valid chars: A-Z, a-z
- No numbers, symbols, whitespaces
- Plaintext message lengths: 1-16 chars
- Shorter messages are padded on the left with ASCII NULL (char code: 0x00)
- All ciphertexts were encrypted with AES-128

Secret Messages to Decipher

These are the secret messages to decipher (crack) in this challenge:

Message ID	Key bits	Ciphertext	Plain text	AES secret key
m8-1	8	0x2aefd1ad57d24bf86ee9019a90896419		
m8-2	8	0x6113cc23d55ffb5e8ddf74654bc1bb9f		
m16-1	16	0x16db1fbad403dc904df58e2bd111e9f9		
m16-2	16	0x965a4221aaf289375d6630b74c146728		
m20-1	20	0x4626b8288582211746f95a58eeb6d7ee		
m20-2	20	0xf0031603cd2279075c4b0d2b3eac0870		
m24-1	24	0x65dcbb9a4e5bd044d911b64ab87f315e		
m24-2	24	0x13ddc51e0434490f026d167948adf877		
m32-1	32	0xdb876859e9a08f4daa56a641ace5da58		
m32-2	32	0x3e38389d9230acac60302d503b28f131		

Rules & Requirements

Deadline -- The competition begins on 11/13 and submission will close on Friday, 11/20 at 12:00pm.

Working mode -- You may work individually or form a team of up to 3 people. If you form a team, please email deapsecure@gmail.com with your team name (no whitespaces, to be used in lieu of “MIDAS ID” in the competition) and the name of the members. You can ask us questions for clarification on deapsecure@gmail.com or the Slack channel used for the workshop.

Submission -- Submit your answer on this form: https://docs.google.com/forms/d/e/1FAIpQLSfn4DNu_f7FTYid_2yEZkAYB1GefaESPeY73EToaz6QP14Q-A/viewform Make a separate submission for each ciphertext. You must also submit the source code and output files as a proof of your solved plaintext and timing (see below). You have two options:

1. Upload your files to your own Google drive folder, and share the folder with

deapsecure@gmail.com . Juges will be able to get your files via this Google account.

2. (Use this way if you are comfortable with UNIX commands.) Create a private subdirectory on Wahab cluster's scratch dir. On Wahab, please issue:

   ```
   mkdir -m 0700 /shared/DeapSECURE/AES-challenge/$USER
   ```

   then use "**`cp`**" or "**`cp -r`**" to create a copy of your files inside this directory. Replace **`$USER`** with your MIDAS ID or team name. For individual submission, **`$USER`** will automatically expand to your MIDAS ID.

What to submit

For any submission option, please organize all the submitted files in the following way: Add a README file (either as Google doc or plain text file) on the root directory of your submission. Please add brief explanation of the following:

1. The crypto library that you use (either aes.py, pycryptodome, or others)
2. How you run the cracking program (do you use notebooks? Or Python script submitted to SLURM?)
3. Parallelization strategy (if applicable)
4. Where you ran your cracking effort (see below)?
5. If applicable, indicate the use specialized computing devices (e.g. GPU)

Please submit all relevant source and output files, as applicable to you: Jupyter notebook and/or Python script and/or C/C++ source files and/or output files and/or the SLURM job script.

The submitted answer needs to have the directory structure like this:

```
ROOTDIR/m8-1/{all your solution files here}

ROOTDIR/m16-1/{all your solution files here}
```

where `ROOTDIR` stands for the root directory of your submission; `m8-1`, `m16-1` and the like stand for the message ID: 8 is the number of unknown key bits, 1 is the message number. If we are not clear about your solution, we will contact you via email for clarification. (We will retain a copy of your submission for our internal assessment of the effectiveness of our training. We will not publish your submission anywhere without your permission.)

Timing -- To be counted for the "**Fastest Decipher**" award, your job script or cracking program must measure the wallclock time taken by the program to crack a ciphertext, defined as the time elapsed from the start to the finish of the program. The elapsed time must be printed to your output file. See hints below.

Where to run -- You can run the key cracker anywhere: on your own PC, on ODU MOVE environment, on your departmental server or VM, or on Turing or Wahab You can use Jupyter notebook, or batch submission. Use CPU, GPU, whatever tech you can get hold of! Have cloud resources? You can use those too. (Our recommendation: submit batch jobs via SLURM on Wahab.)

Restrictions -- You must honestly write your own program. You are free to search for information on the Internet, but you are **<u>NOT</u>** allowed to use any existing ready-made code (such as John the Ripper or code written by friends not on your team). The only external modules/libraries you can use are the AES library with its encrypt/decrypt routines. See hints below for suggested choices.

(End of required reading. Next pages contain resources to help you.)

Basic How-To & Hints

Brute force decryption of a message

Messages and keys are given/expected in hexadecimal representation:

```
0x0F = 15
0xF0 = 15 * 16 = 240
0xFF = 255
```

Example secret message:

```
0xd3bbd04550497f943f6bf4c9e2291993
```

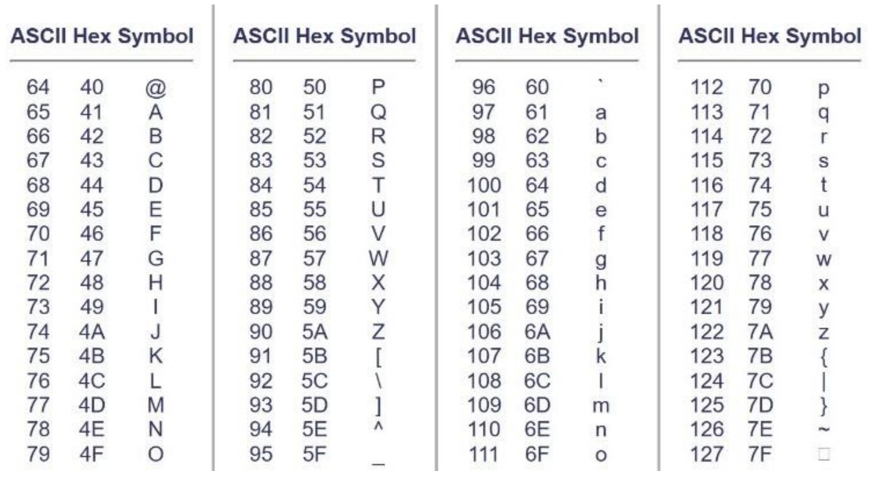
In the first challenge, for a key with one-byte unknown, the key looks like:

```
0x000000000000000000000000000000??
```

Combinations to try are 0, 1, 2, ...255 (total 256 numbers to try).

Hints

- Use the `for` loop construct to brute-force the key trials.
- Validating the decrypted messages will require you to validate every character in the messages. Some functions in the Python standard library can make your life easier. See Python documentation on bytes and bytes-like objects .
- You are strongly encouraged to use the parallel computation techniques to break the ciphertexts in the shortest amount of time.
- The following standard ASCII table was used to encode string plaintext in the two challenges, where “ASCII” is the decimal number, “Hex” is the hexadecimal format, and “Symbol” is the corresponding symbol. For example, the symbol “A” is encoded by 65 in decimal or 0x41 in hexadecimal, “a” by 97 (0x61) and so on and so forth. **You can just consider the symbols from “A” to “Z” and “a” to “z” with (hex value 0x41 - 0x5A, and from 0x61 to 0x7A)**.



ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol	ASCII	Hex	Symbol
64	40	@	80	50	P	96	60	`	112	70	p
65	41	A	81	51	Q	97	61	a	113	71	q
66	42	B	82	52	R	98	62	b	114	72	r
67	43	C	83	53	S	99	63	c	115	73	s
68	44	D	84	54	T	100	64	d	116	74	t
69	45	E	85	55	U	101	65	e	117	75	u
70	46	F	86	56	V	102	66	f	118	76	v
71	47	G	87	57	W	103	67	g	119	77	w
72	48	H	88	58	X	104	68	h	120	78	x
73	49	I	89	59	Y	105	69	i	121	79	y
74	4A	J	90	5A	Z	106	6A	j	122	7A	z
75	4B	K	91	5B	[	107	6B	k	123	7B	{
76	4C	L	92	5C	\	108	6C	l	124	7C	|
77	4D	M	93	5D	]	109	6D	m	125	7D	}
78	4E	N	94	5E	^	110	6E	n	126	7E	~
79	4F	O	95	5F	_	111	6F	o	127	7F	□

source from https://ascii.cl/

- Timing your code run time:
    - As a starting point for your job (SLURM) script, please see the following file in your "module-hpc":
        - Spam_bash/templates/year1998.template
    - Measuring time (C/C++): https://www.geeksforgeeks.org/measure-execution-time-with-high-precision-in-c-c/
    - Measuring time (Python): https://www.geeksforgeeks.org/python-measure-time-taken-by-program-to-execute/

Performance & Choice of Libraries

Here are some libraries you can use for this competition. You are also free to use other AES crypto libraries implemented in your preferred programming language.

Performance & parallelization really matter to be able to crack the most difficult messages!

Choice 1: AES library in pure Python

https://github.com/bozhu/AES-Python

This is the "aes" module that you've been using in the workshop. Please refer back to your notebook #2 from the workshop (download: notebook, HTML).

Choice 2: Faster implementation: PyCrypto(dome)

https://www.pycryptodome.org/en/latest/src/introduction.html

https://www.pycryptodome.org/en/latest/src/examples.html

(use ECB mode because we only have one block to encrypt.)

PyCryptodome is a more up-to-date fork of the original PyCrypto library. It is available as the "Crypto" Python module. On Wahab we have this library available as part of the DeapSECURE suite.

Hint: This will run at least 30x faster than aes.py !!

Choice 3: Tiny AES in C

https://github.com/kokke/tiny-AES-c

Even faster implementation, but requiring complete programming in C or C++.

Hints:

- Use the AES_ECB_encrypt and AES_ECB_decrypt functions.
- One of the team members coded the whole cracker code using this library in under 250 lines in a "clean" fashion, so this is not too big a stretch.