

License Statement

This is a Python version of Chacha20. It is a modification of a program posted by Jack O'Connor at https://github.com/oconnor663/pure_python_salsa_chacha. The unmodified programs are in the public domain. The modifications as provided by us are NOT in the public domain and all rights are owned by LCIP JV. Any use in operational application is prohibited. We used Mr. O'Connor's publicly posted programs because 1) they work 2) they are well structured and easy to modify and 3) they correctly process test vectors. As such they are a strong basis to experiment with.

The application of the FLTed function and other functional transformation like radix-n transformation and use of n-state involution functions for $n=2^k$ not being additions over $GF(2^k)$ in characteristic in cryptography is protected by issued and pending US patents. No license is provided for application of the claimed subject matter in these IP cases. Contact us at info@labcyfer.com for any further information.

Warning

The effect of the functional transformations is seemingly a simple one. Don't be deceived. We are talking about factorial levels of modifications. For $n=256$ we are talking about a factor 10^{500} . The radix-256 transformation provides a level of uncertainty of a factor greater than $10^{1,000}$. Believe us, there is no way that you can remember changes you make in functions like sc256, sn256 and car256. If you do, please save the tables under their own names. We have absolutely no way to retrieve lost or forgotten parameters. You are completely on your own on this.

How to run the programs under Python!

These programs replace bitwise XORing of words of 32-bits in the state array with a 256-state function sc256 or sn256 (different from each other) provided as lookup table.

1) unzip ChaCha20FLT.zip is its own folder, for instance 'chachafltpython'

2) open the .py files, for instance in Notepad and make sure you have installed the required libraries

3) open CMD terminal and go to the folder of the programs

4) to run the single test:

python testcha.py

In this program bitwise XOR of bytes is replaced by lookup table addition over $GF(256)$ (which are identical). It runs a single test on the ChaCha20 generated keystream. It shows that 'found' is same as 'expected.'

5) run 4 tests

python test_chacha20enc.py

This program runs 4 test vectors with the same replacement using the lookup table sc256.

6) enter and run a plaintext encryption/decryption

python test_chacha20encrypt1.py

This requires to input plaintext. It also decrypts and shows the correctly recovered plaintext

7) use the FLTed function sn256 which is different from 'sc256'

```
python test_chacha20encrypt1snsn.py
```

This program applies lookup table sn256 different from sc256, which is an addition over GF(256) but is NOT equivalent to XORing of 8-bit words. Outputs are also provide in hex-strings for comparison.

8) use the function sc256 in encryption and FLTed function sn256 in decryption'

```
python test_chacha20encrypt1scsn.py
```

The recovered plaintext is different from the actual plaintext. One may compare the hex strings of plaintext and recover text.

An error will occur and is expected, as the recovered plaintext may have unprintable characters

Feel free to mix and match programs if you are comfortable with it.

Several additional different lookup tables are included: sn256a, sn256b, sn256c and sn256d.

You can modify in for instance *pure_chacha20fltsn.py*

```
def chacha20_permute(block):
```

```
    data_funrev = scipy.io.loadmat('sn256.mat')
```

```
    funrev = data_funrev['sn256']
```

Replace sn256 with desired replacement like sn256a:

```
def chacha20_permute(block):
```

```
    data_funrev = scipy.io.loadmat('sn256a.mat')
```

```
    funrev = data_funrev['sn256a']
```

November 18, 2024 LCIP JV