

## 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](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 radix-256 functional transformation in combination with a look-up table for an addition modulo-256. The modification is subject of a pending US patent. No license is provided for application of the claimed subject matter in these IP cases. Contact us at [info@labcyfer.com](mailto: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 of at least  $10^{15,000}$ . This is an insanely large number and it is NOT a typo. Believe us, there is no way that you can remember changes you make in the table car256. 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 addition modulo- $2^{32}$  with a radix-256 addition of words of 4 bytes. The transformation in this example is applied in the 256-state carry table.

**1) unzip ChaCha20rad.zip is its own folder, for instance 'chacharadpython'**

**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 4 test\_vector tests:**

*python test\_chacha20rad.py*

This program has replaced the add32() with a ripple scheme using lookup tables sc256mod and car256mod. This is identical to addition mod- $2^{32}$ . It should output a series of 'good testx' messages.

**6) enter and run a plaintext encryption/decryption, 'car256mod' is replaced by 'car256'**

*python test\_chacha20radx.py (see added x to name)*

This requires to input plaintext. It also decrypts and shows the recovered plaintext. For comparison, the plaintext and recovered text are shown as hex strings. The recovered text should be same as plaintext

**7) encrypt with modified chacha20 and decrypt with standard chacha20**

*python test\_chacha20radnx.py*

This program encrypts with chacha20radx and decrypts with chacha20rad. It generates unrecoverable text. All key and iv inputs are the same, but the transformation prevents correct decryption.

**November 18, 2024, LCIP JV**