Leveraging Graphene with Intel-SGX to run applications in hardware based secure Enclaves

# 

# 

[**Graphene SGX:**](#_heading=h.3znysh7) **[2](#_heading=h.3znysh7)**

[Manifest syntax](#_heading=h.2et92p0) [2](#_heading=h.2et92p0)

[**Intel-SGX:**](#_heading=h.tyjcwt) **[2](#_heading=h.tyjcwt)**

[**Installation of graphene:**](#_heading=h.3dy6vkm) **[3](#_heading=h.3dy6vkm)**

[**Results:**](#_heading=h.1t3h5sf) **4**

[Simple](#_heading=h.4d34og8) [4](#_heading=h.4d34og8)

[Graphene](#_heading=h.2s8eyo1) [4](#_heading=h.2s8eyo1)

[Graphene-SGX](#_heading=h.17dp8vu) [5](#_heading=h.17dp8vu)

[Performance](#_heading=h.3rdcrjn) [6](#_heading=h.3rdcrjn)

[**System Scheme Architecture:**](#_heading=h.26in1rg) **[7](#_heading=h.26in1rg)**

[Attack Surface:](#_heading=h.lnxbz9) [7](#_heading=h.lnxbz9)

[**Exiting files in folder:**](#_heading=h.35nkun2) **9**

[Specific Commands](#_heading=h.1ksv4uv) [10](#_heading=h.1ksv4uv)

[**Useful Link | Reference:**](#_heading=h.44sinio) **11**

# Graphene SGX:

Graphene [[[1]](#footnote-0)] is a lightweight guest OS, designed to run a single Linux application with minimal host requirements. Graphene can run applications in an isolated environment with benefits comparable to running a complete OS in a virtual machine – including guest customization, ease of porting to different host OSes, and process migration.

Graphene supports running Linux applications using the Intel SGX (Software Guard Extensions) technology (we call this version Graphene-SGX). With Intel SGX, applications are secured in hardware-encrypted memory regions (called SGX enclaves). SGX protects code and data in the enclave against privileged software attacks and against physical attacks on the hardware off the CPU package (e.g., cold-boot attacks on RAM). Graphene is able to run unmodified applications inside SGX enclaves, without the toll of manually porting the application to the SGX environment.

## Manifest syntax

A manifest file is an application-specific configuration text file that specifies the environment and resources for running an application inside Graphene. A manifest file contains key-value pairs (as well as more complicated table and array objects) in the TOML syntax. For the details of the TOML syntax, see the official documentation.

# Intel-SGX:

Intel® Software Guard Extensions [**[[2]](#footnote-1)**] (Intel® SGX) is a set of instructions that increases the security of application code and data, giving them more protection from disclosure or modification. Developers can partition sensitive information into enclaves, which are areas of execution in memory with more security protection.

For details to understand of Intel-SGX kindly follow these links[[[3]](#footnote-2)].

# Installation of graphene:

To install graphene with Intel-SGX support kindly follow this guide[[[4]](#footnote-3)]. Major steps for this process will be as:

1. Required packages
2. 2b. Install the Graphene FSGSBASE driver (not for production)
3. Generate signing keys
4. Install the Intel SGX driver and SDK/PSW
5. Building

**NOTE**: Kindly do not follow **2a. Install the Linux kernel patched with FSGSBASE** instead use step 2 as above. To follow 2a hardware kernel level changes are required which could take a lot of space in hard disk, also it will take a lot of time. So the best option is to ignore it, but later we will need to come back to this step for product usage for application.

After following all steps graphene might not be working as there are some errors in online guidance. To properly run graphene with SGX it has to be noted that at the end we will have to use thee two commands as

sudo insmod gsgx.ko ------------------- 1

To run this command you will have to go in the directory where graphene-SGX driver is installed. In this command, insert mode is used to insert a driver in the kernel.

If you find some problem then we need to install Intel SGX driver first then after that install Graphene driver then insert this in kernel. **OR** in other words this command should be used after intel-SGX driver installation.

After this below command (2) should be used in terminal. We can use (1) and (2) command in any order. (we can use command 2 first and command 1 in second order this does not make any error)

sudo sysctl vm.mmap\_min\_addr=0 ------------------- 2.

**NOTE**:

You will have to run these two commands, every time when you start your computer again. (restart, after shutdown etc).

To understand few issue in installation read this is required [[[5]](#footnote-4)]. Also, just like this kindly check issues link on GitHub if you get any error. Solved issues are very useful to understand possible error you might encounter.

# Results:

We have implemented caesar-cipher code for encryption as well as for decryption purpose. We used data from a text file called test.txt then used this data with some input data from the terminal for encryption and decryption purpose. Both encrypted and decrypted data is placed in the same directory.

All implemented code is well tested and can be easily converted to RSA based encryption and with the help of python3 libraries we can also easily implement ECC code based on SGX.

Our main code of caesar-cipher is implemented in three scenarios as:

* Simple
* Graphene
* Graphene with SGX support

Same code is implemented in all three scenarios and evaluation is measures based on number of CPU cycles utilized by these cases. *Perf* tool is leveraged for measuring performance relevant states.

## Simple

In simple method implementation is carried using *gcc* compiler without any other helping tool. As simple *gcc* compiler generate an output which has the least dependencies that is why least CPU cycles are used in this case.

## Graphene

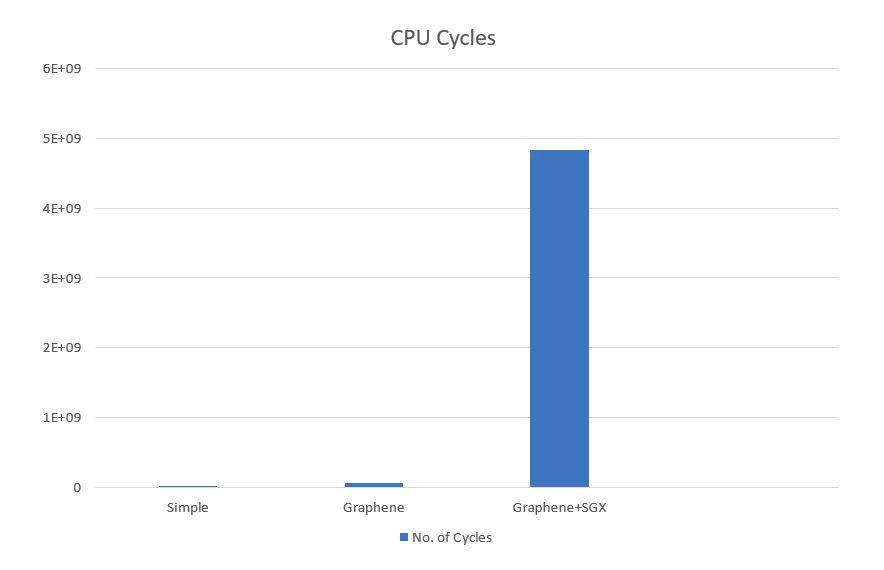
In Graphene based implementation, our code is run in virtual space created by graphene libraries and for this purpose glibc is used and some default graphene based modified glibc libraries are used, all these dependencies and virtual environment created take a lot of CPU cycles which are again measured by *perf* tool.

## Graphene-SGX

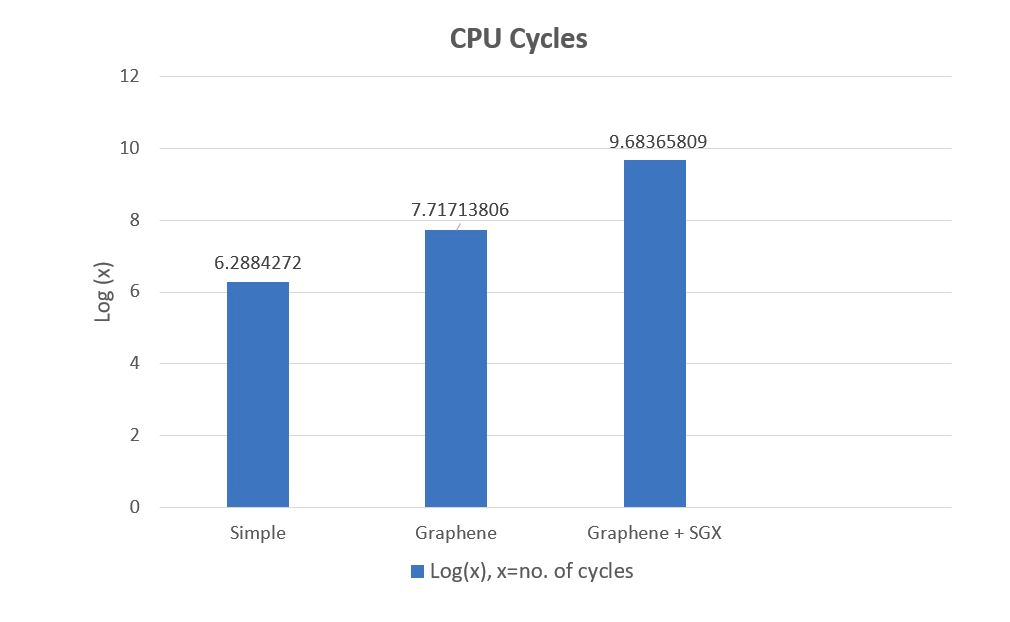
In the last case, we created an enclave in real SGX hardware and implemented our code inside Enclave. In enclave coding graphene libraries are used with minimum modified coding requirements. Graphene translated SGX side to glibc and glibc side to SGX coding. For this purpose in manifest file we have declared all the dependencies and trusted files to be required to run our code inside Enclave. Due to a lot of more dependencies and graphene translation. CPU cycles used in this case are much more than our previous cases. The Major reason for this, we can observe for this is because the initial SGX enclave graphene has to process a lot of libraries before actual code running due to which load on CPU increase exponentially. This load does not increase with more code as this only depends on number of dependencies to be initialized for Enclave to trust, so in a big real application CPU cycles different will be different from our results in which there is a huge difference as implemented code is simple and major overhead is only due to preloading of libraries for SGX Enclave.

## Performance

A bar graph is plotted where the number of CPU cycles are indicated against each technique.



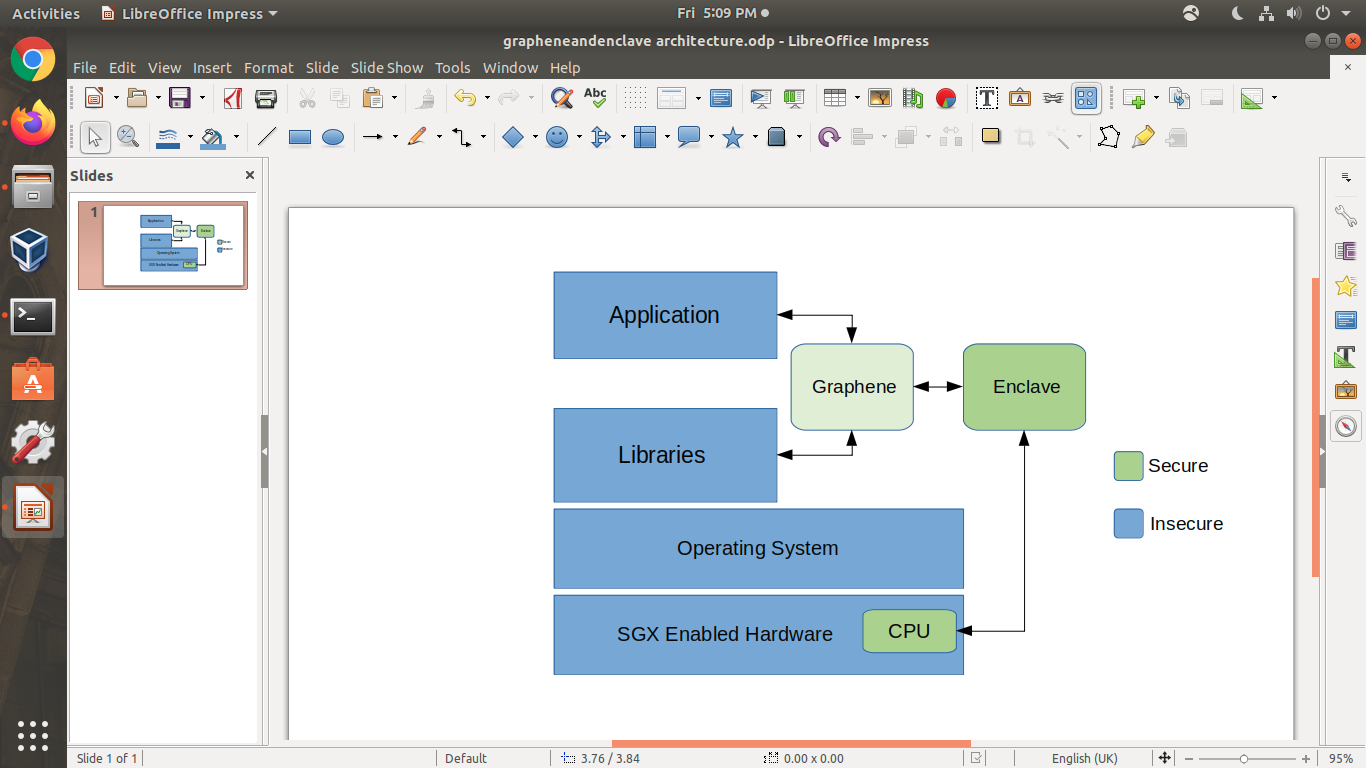
As this graph does not provide good visual results, so to get better bar graph visual results we have used log(x) to visualize the number of CPU cycles in terms of log, here x shows the number of CPU cycles. This graph has good scaled output than previous.



# System Scheme Architecture:

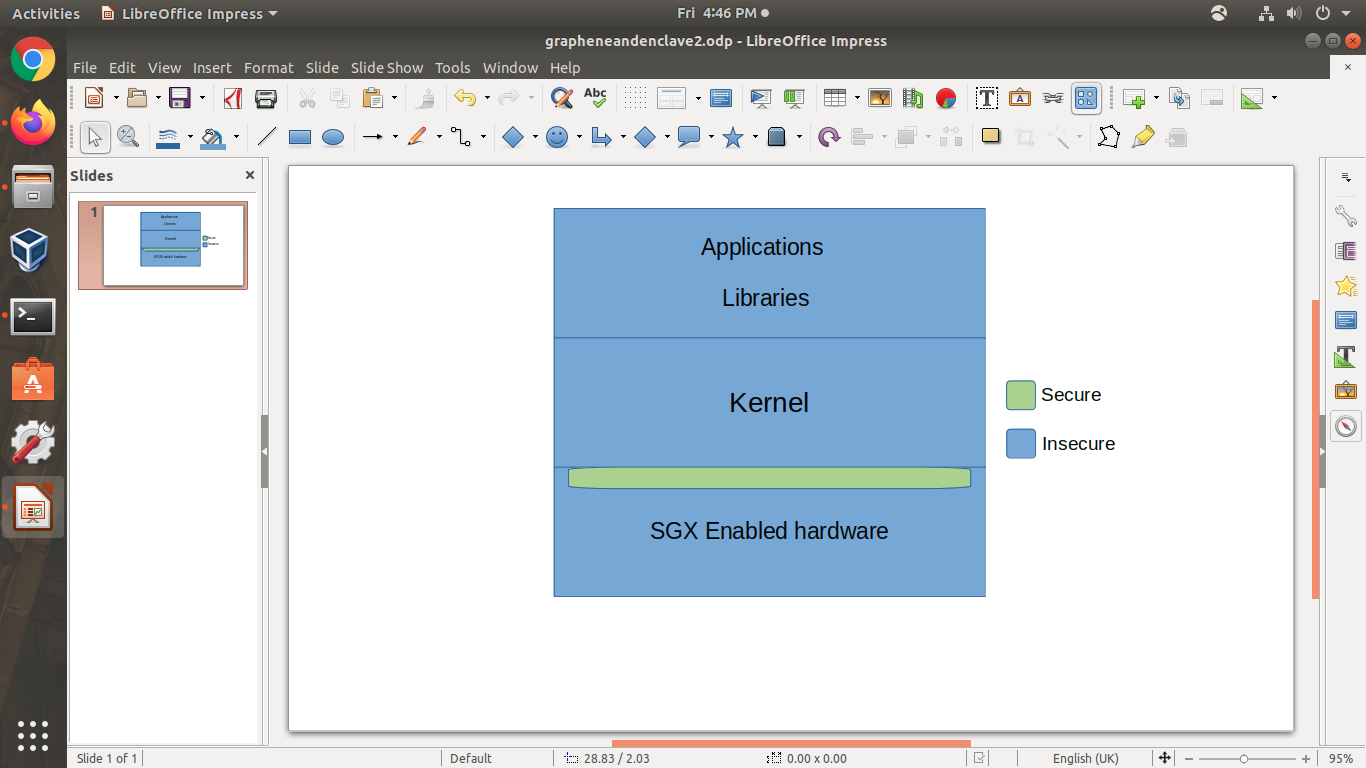
In our system scheme, Graphene is acting as a translator library where application code and dependencies are being translated to SGX based coding by help the help of Graphene. Graphene preload libraries and trusted files and load in SGX Enclave at the time of Enclave creating. Every hash output from these files and graphene based SGX-manifest file hash are also added to Enclave’s trusted data report. This ensures that every trusted file remain unmodified, and whenever someone modifies any of these file and as hash is changed, so the final resultant hash is also changed due to which Enclave creating call is failed. This feature is very crucial for ensuring data integrity in our SGX based implementation.

Below is the simple architecture for our implementation where top to lower layers of a computer are shown with data flows of implementation. Green colour represent trusted portion and blue represent potential attack surface by others. Here Graphene is light green which mean graphene is secure but as it somewhat depends on operating system and user initially that is why there are some possibilities for attack.

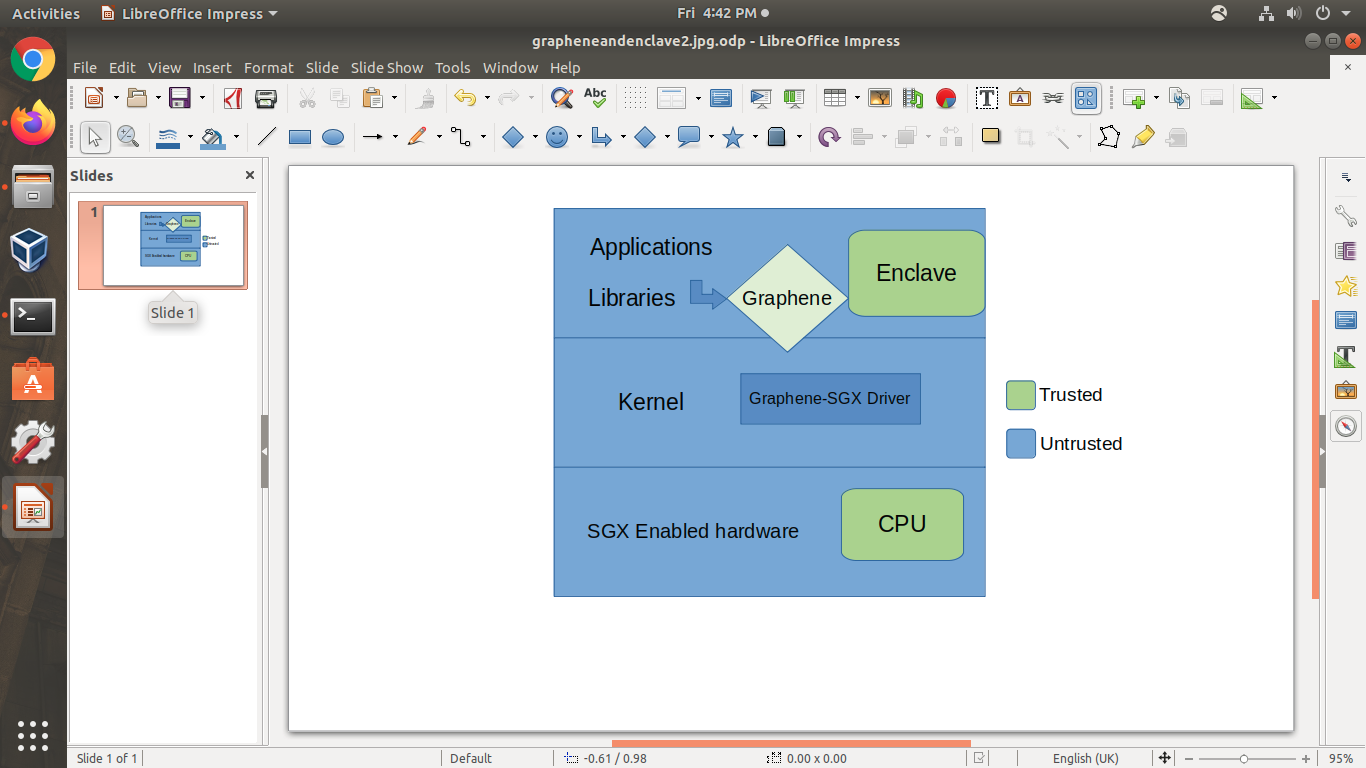


## Attack Surface:

Running applications in normal mode with our SGX support cannot provide reduced attach surface. An attacker can attack by different sources to get his targeted information for example application APIs, libraries, operating system or even hardware on which the application is running could be manipulated for attach.



By this attack surface can be easily reduced by help of SGX enabled hardware where we execute our secrets inside an Enclave which is secure from external attacks. This way an attacker cannot get information about the process of the application running inside Enclave by previous mentioned means. Here the attack surface is greatly reducing too only to Enclave and CPU as every information going outside CPU is being encrypted by hardware key building by intel.



# Exiting files in folder:

Some existing files in the drive folder are as;

* **Dev**
* **Dev\_manifest**

There are some other files too, but only the above two are actually required to run code, the other will be updated when you run command

make SGX=1 sgx-tokens

After this below command is to be used to run code:

SGX=1 ./pal\_loader dev

In **dev.c** file, it contains a code for default directory printing, but we have used our own code inside dev.c so that we do not have to modify manifest, make and pal.loader files. The Purpose of C language code in dev.c is to:

* Read from a file
* Store data in a variable
* Encrypt data read from file
* Save encrypted data
* Decrypted data after reading encrypted file.

In code files are text formatted but other formates will work fine too. In code RSA based technique is used and here only one number for encryption is used, but we can use full RSA based three numbers too, but our main goal is to encrypt and decrypt data using elliptical curve cryptography (**ECC**) technique.

Read operation will consist on reading two file one for provided key and the other for provided data file. Then based on the provided key we have to encrypt data inside enclave, to we will make a **Hash** out of it which will be used for verifications later.

Provided key can be hardcoded inside our code for better security purpose, as **dev.c** file can also be deleted once an Enclave is created and other users cannot read code inside enclave.

**NOTE**: Kindly view all comments for understanding code

\*Currently, work to save keys inside Enclave is required which is very important to utilize our work in **PKIchain** project.

## Specific Commands

* To provide an Intel-SGX driver path for Graphene below command is used. This command is crucial to pick Intel-SGX driver otherwise you will face to *make* compilation e error for graphene due to incorrect path or NULL path set. Please note this command is specific to the computer in which this setup is made, if the user is in some other computer then kindly change this command according to that.

ISGX\_DRIVER\_PATH="/home/maryam/Documents/SGX-real/linux-sgx-driver/"

* During installation of SGX-SDK, environment setup is important via SDK before compiling a code e.g.

source ${sgx-sdk-install-path}/environment

In our desktop this command is used like below

source /home/maryam/Documents/SGX-real/linux-sgx/linux/installer/bin/sgxsdk/environment

To make it work with SGX kindly do not install graphene-SGX driver first, there is error in build documents due to misrepresenting of driver for SDK for kernel.

* When you have installed all dependencies and drivers for Graphene, you can make graphene with and without SGX at the same time, this will create different files for both, this is useful if we need to test a code with and without SGX.
  + Without SGX command is as make
  + With SGX command is as make SGX=1
  + Remember if there come any error in compilation then you must first fully clean compiled graphene before making again, other you will face the same error even if the error issue if resolved. To fully clean make data use command as

make disclean or make SGX=1 disclean

* add sign key in pal folder from graphene main using

openssl genrsa -3 -out Pal/src/host/Linux-SGX/signer/enclave-key.pem 3072

Otherwise, programs will not run as above command generate key in signer folder, if the key generated is not inside signer folder then you have to set the key path for key or copy key to signer folder, where later is better and suggested due to being a default choice.

# Useful Link | Reference:

All useful links and references are provided in **footnotes**.

1. https://graphene.readthedocs.io/en/latest/index.html [↑](#footnote-ref-0)
2. https://software.intel.com/content/www/us/en/develop/topics/software-guard-extensions.html [↑](#footnote-ref-1)
3. https://graphene.readthedocs.io/en/latest/sgx-intro.html [↑](#footnote-ref-2)
4. https://graphene.readthedocs.io/en/latest/building.html [↑](#footnote-ref-3)
5. https://github.com/oscarlab/graphene/issues/2095 [↑](#footnote-ref-4)