Design and Implementation of a Parallel Computing Framework for Efficient and Secure Lattice-Based Key Exchange Protocols

In open network sender and receiver can exchange data but this data can be infer or hack by malicious user and there is no direct way to detect such infer and to overcome from such issue many key exchange algorithms was introduced such as Diffie Hellman, Lattice and many more. In all algorithms Lattice is consider to be more secure.

Sender and receiver can make use of this Lattice key exchange and verification algorithm over NTRU protocol. Lattice allow sender to generate Secret Key and then sign message (generate signature of message using secret key) using secret key and send both Message and signature to receiver. Receiver will generate private key using Lattice algorithm and then regenerate signature on received message and then verify received signature and generated signature and if message is same then both signatures will have same values and no hacking will be detected. If message changed by hacker then it will result into different signatures and verification will be failed.

So by using above techniques we can securely communicate over network but Lattice key generation require heavy computation time and to efficiently manage computation we are applying OPENMP technique on Lattice algorithm to distribute key generation and verification task between multiple parallel processes which can lead to lower computation time and make Lattice efficient.

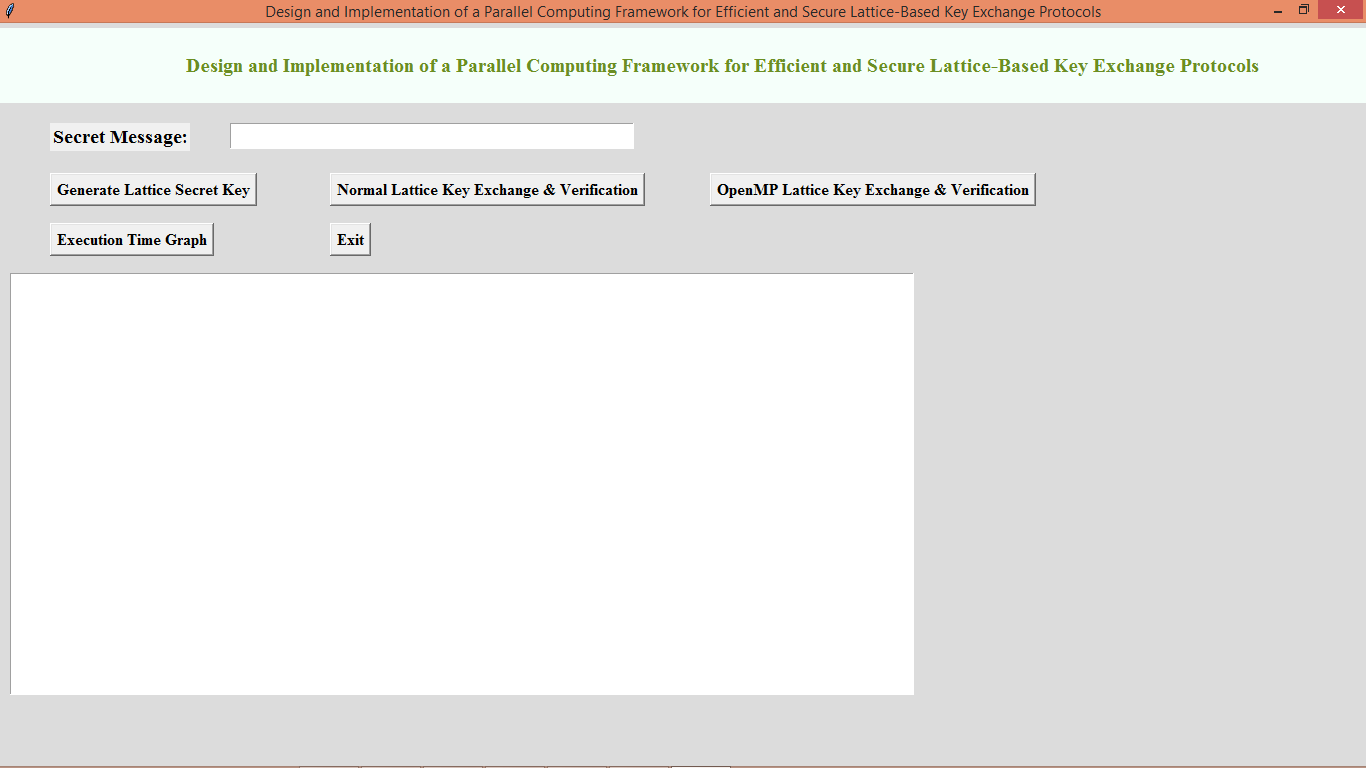
To evaluate performance we are implementing message signing simulation using normal and parallel Lattice Key Exchange algorithm and then compare execution time between them.

To implement this project we have designed following modules

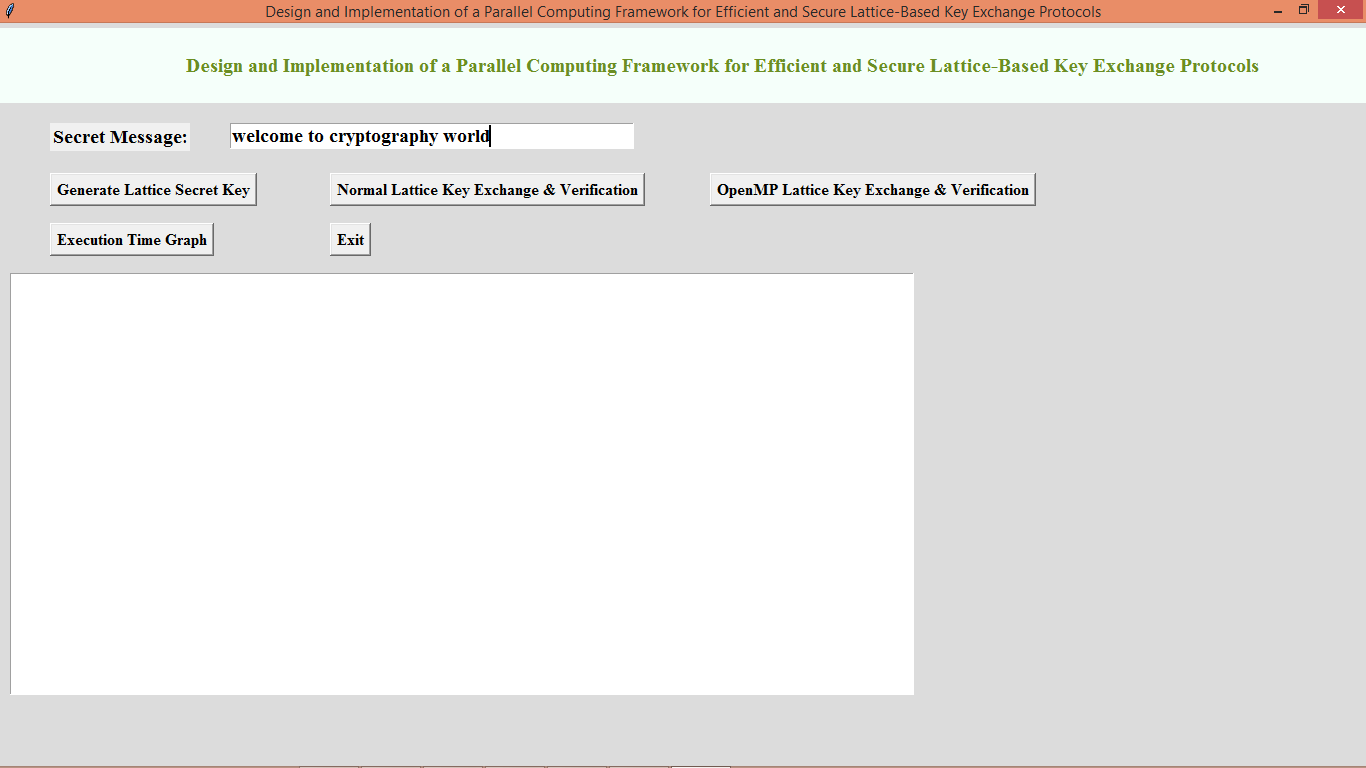
1. Generate Lattice Secret Key: using this module we will creating Lattice secret key and then distribute between sender and receiver
2. Normal Lattice Key Exchange & Verification: using this module sender can enter some message and then generate signature on that message using secret key and then send to receiver. Here Lattice will get executed as single thread normal process and then calculate execution time
3. OpenMP Lattice Key Exchange & Verification: using this module sender can enter some message and then generate signature on that message using secret key and then send to receiver. Here Lattice will get executed as OPENMP multi-process threads and then calculate execution time
4. Execution Time Graph: using this module we will plot execution time of normal and parallel Lattice key exchange algorithm.

SCREEN SHOTS

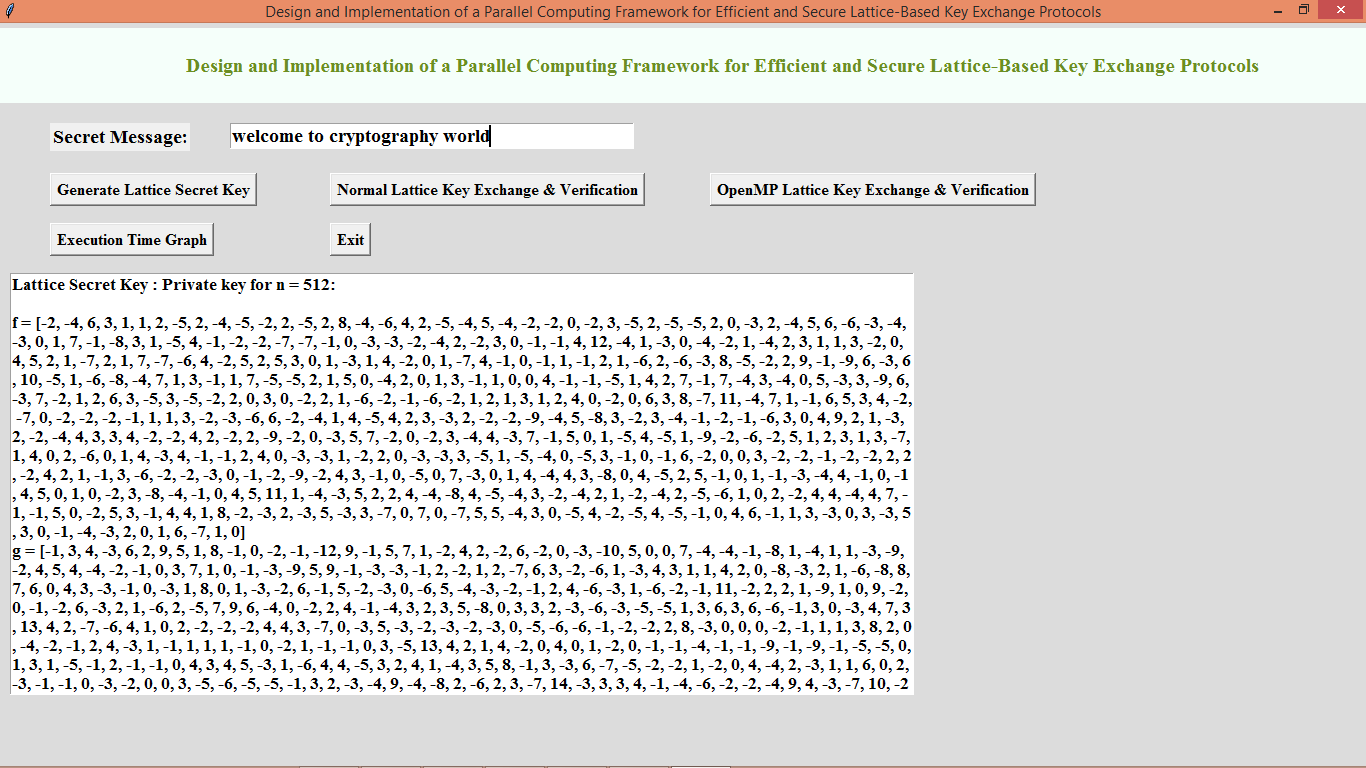
To run project double click on ‘run.bat’ file to get below screen



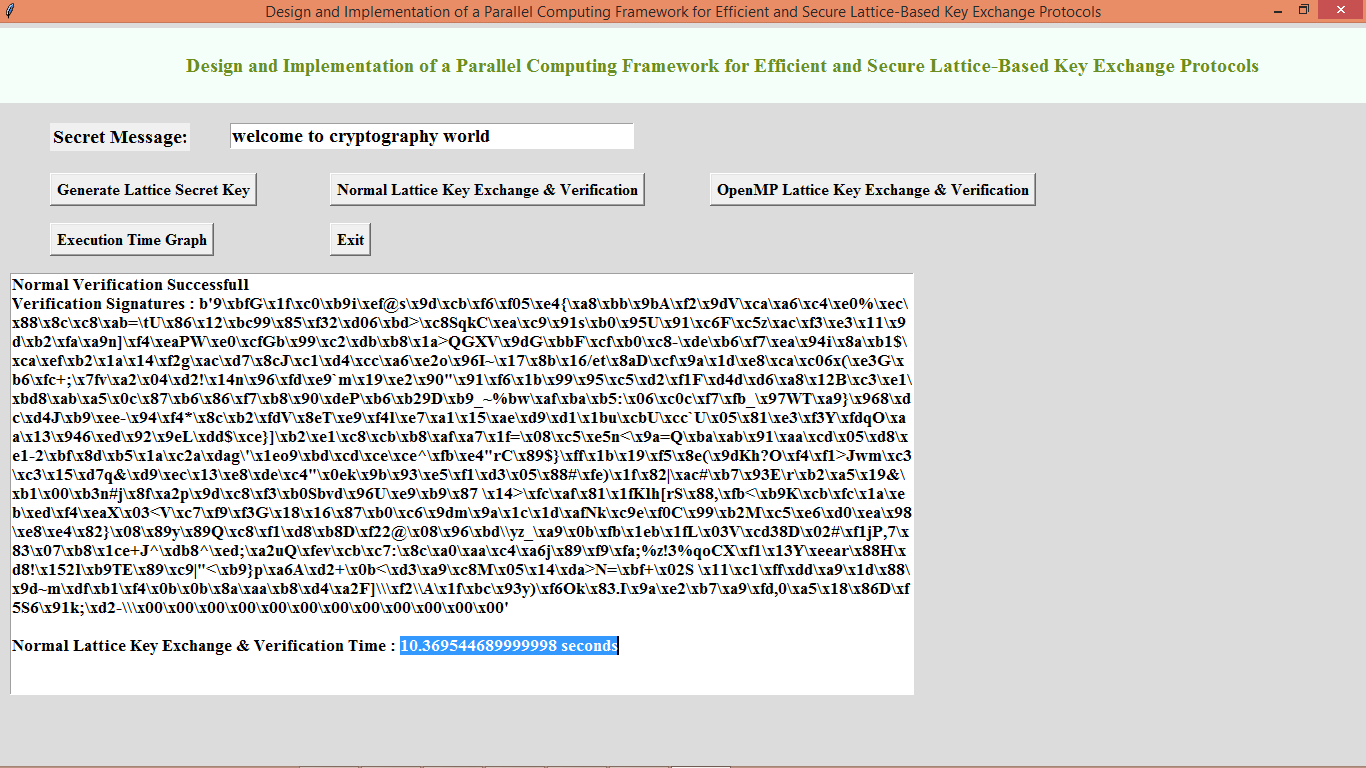
In above screen enter some message and then click on ‘Generate Lattice Secret Key’ to generate secret key



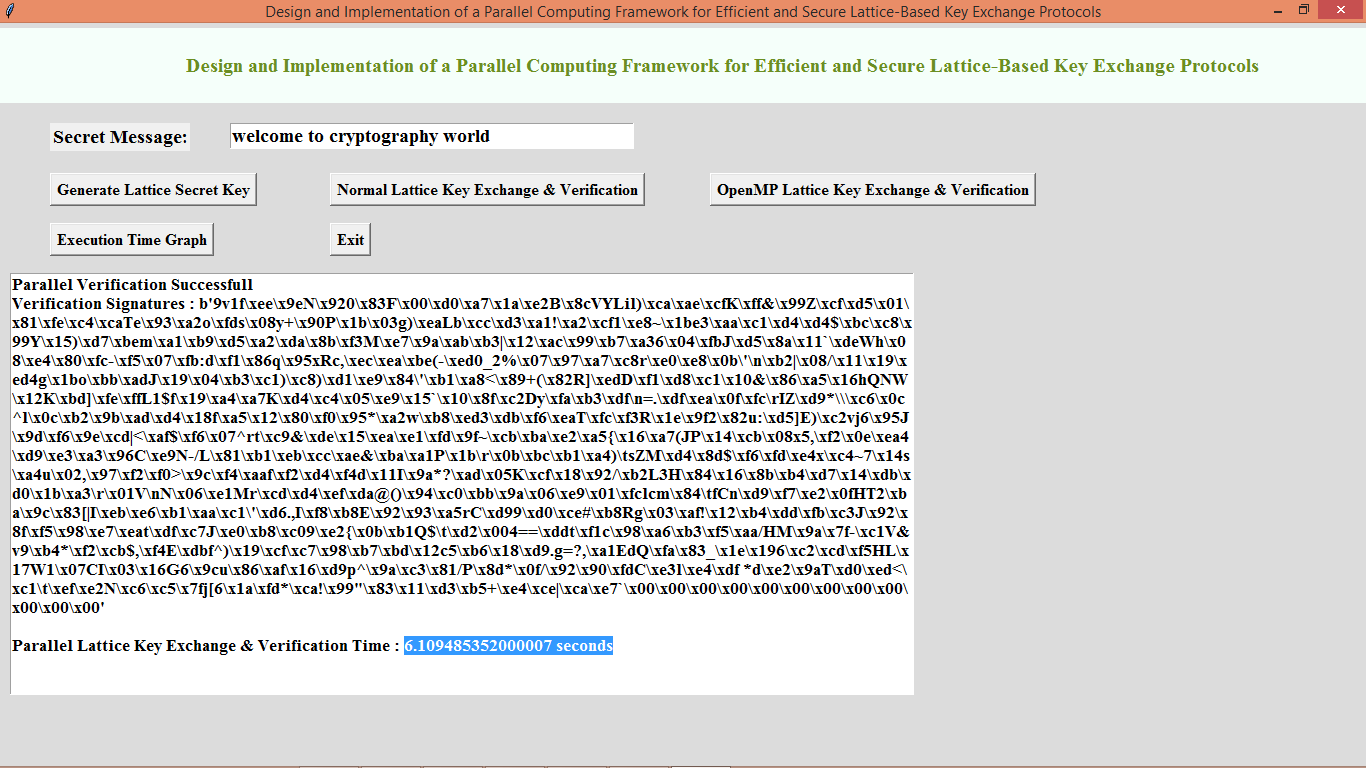
In above screen i entered some message and then click on ‘Generate Lattice Secret Key’ button to generate key and get below output



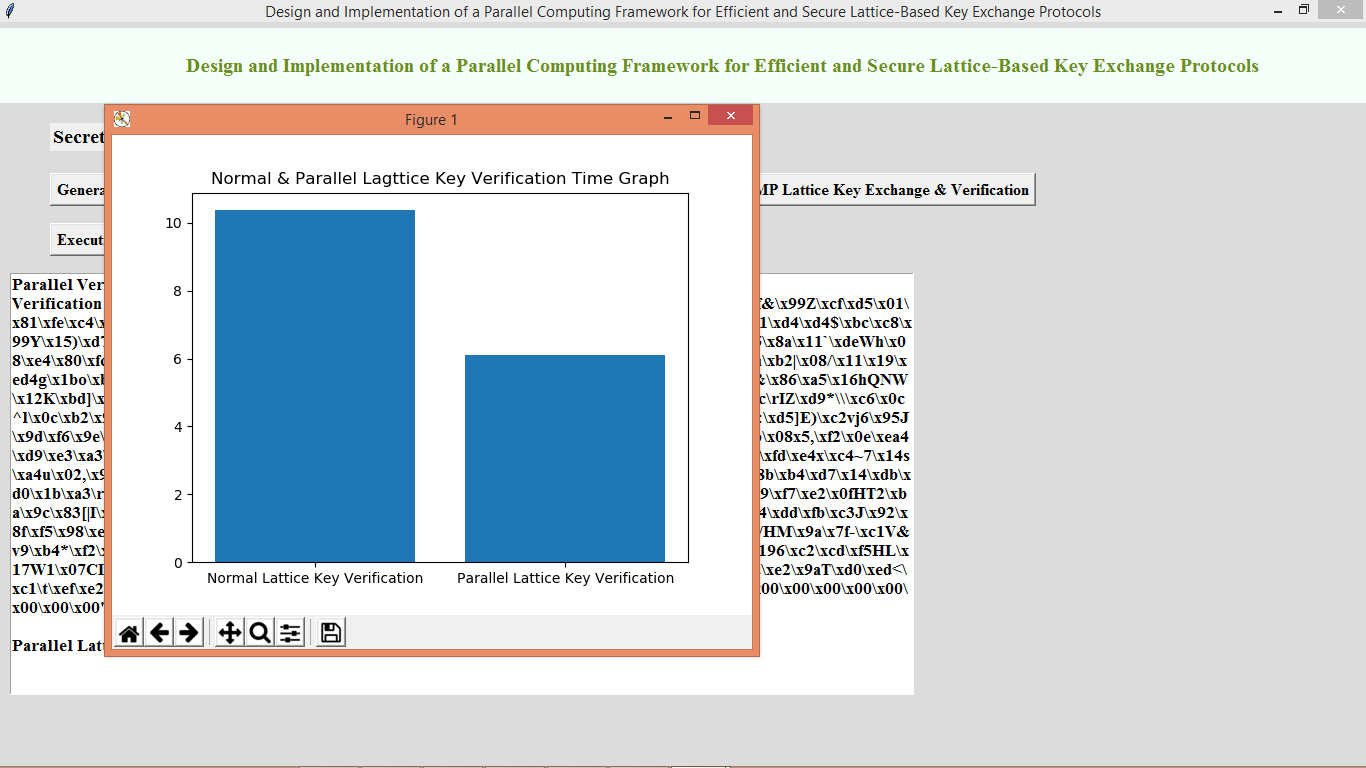
In above screen secret key is generated and we can see key is very complicated and nobody can guess or hack that key and now click on ‘Normal Lattice Key Exchange & Verification’ button to sign message with above key and then perform verification and get below output



In above text area message is send to receiver and then in first line we can see message is verified successfully and in next line we can see generated signatures on message and in last line we can see Lattice took 10 seconds with normal single thread process and now click on ‘OpenMP Lattice Key Exchange & Verification’ button to generate and verify keys using OPENMP and get below output



In above screen to generate signature and to verify keys OPENMP parallel process took 6 seconds so we can say with Parallel technique we can enhance lattice performance. You can test with any other message and now click on ‘Execution Time Graph’ button to get below graph



In above graph x-axis represents technique names and y-axis represents computation time in seconds and in both algorithms OPENMP took less execution time.