Benchmarking the Diffie-Hellman key exchange

Kenny Lee, Alex Darwish, Nathan Edmon, and Taehyuan Yoon

***Abstract*—There are often malicious actors on untrusted data networks looking for sensitive data from unsuspecting users. Sensitive data needs to be kept protected from these malicious actors. The Diffie Hellman key exchange provides a way to secure communication between users on untrusted networks through the exchange of keys on a public network. Of course, the importance of resource usage of the Diffie Hellman key exchange is a very important benchmark to look to see if systems like these can provide adequate security while using fewer resources.**

I. Introduction

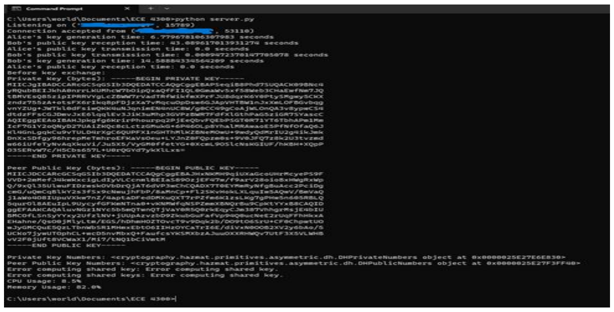
We aim to look into the resource usage of the Diffie-Hellman key exchange on a small scale to see how these systems can be scaled up to provide security on a massive scale. We are looking into specifically two different implementations, one purely through software and another purely through hardware.

II. Methodologies

In our methodologies, we looked into two different implementations of the Diffie-Hellman Algorithm. We implemented a Python and Verilog version. We specifically implemented these two versions as Python is a low barrier to entry when it comes to implementing algorithms and Verilog as a hardware implementation also provides a baseline for a purely software comparison.

III. Results

In our Python implementation, we can see the key exchange take place. First Alice generates her private and public keys with Alice taking 6.78 seconds to compute her keys. Bob receives Alice's public key taking 43.09 seconds to receive. Alice's public key takes 0.0 seconds to transmit to Bob, indicating that network reception is fast. Bob takes 0.000947 seconds to transmit his public key to Alice. Now Bob generates his public and private key taking 14.589 seconds. Alice receives Bob's public key in 0.0 seconds also indicating that network reception is fast. Before our key exchange takes place, we display both Alice's and Bob's public and private keys along with their private key and peer public key numbers. We did receive an error when computing the shared keys indicating an issue in the key exchange process. When running this process we see our CPU Usage is 8.5% with memory usage being at 82%. In our FPGA implementation, we created a key exchange locally on the A7 100t FPGA. Our goal is to view the Diffie-Hellman key exchange resource utilization, look up tables, and memory usage. We are primarily looking into how this system can be scaled into larger platforms or



```
Command Prompt
C:\Users\world\Documents\ECE 4300>python server.py
Listening on ('[redacted]', 15789)
Connection accepted from ([redacted], 53110)
Alice's key generation time: 6.779678106307983 seconds
Bob's public key reception time: 43.089617013931274 seconds
Alice's public key transmission time: 0.0 seconds
Bob's public key transmission time: 0.0009472370147705078 seconds
Bob's key generation time: 14.588843345642089 seconds
Alice's public key reception time: 0.0 seconds
Before key exchange:
Private Key (bytes): -----BEGIN PRIVATE KEY-----
MIICJgIBADCCARcGCSqGSIb3DQEDATCCAQgCggEBAPSeqi88Phd7SUQACK098Nc4
yRQubBEIJkhA0nrrLKUMhcW7bOipQxaQfFI1QL0GmaWvSxfS8Web3CHaEwfNm7JQ
tBMVEsQ85zipIPRRVYgLcZ8WW7rVadTRfWikfmXPrFJU8dqrK6Y0PtySMgmy5CKX
zndz755zA+otsFX6rIkq8pFDjzXa7vMqcuOpDsm6GJApVHTBW1nJxXmLOF8Gvbqg
vnYZUg+JWTkl0dFsimQKK4uNJqnimEN4nUC8W/g0CC49gCoAjWLOnQA3v8ypwCS4
dtdzFFsCGJDmvJxE6lqqlEv3JiK3uMhp3GVPzBWR7FdfXlGthPaG5ziGR7SYascC
AQIEggEEAoIBAHJpkgfg6KrirPhourpq2PjEeQbvFQEbPSGT0R71YT6TbhAPe1Mm
IcF7G1Y2oQNyD27UAiZkQcBcLctzGMukG+6P06OLp8YhalMRAmaoE5PfNfOfaQ6J
Kl4GnLgqkCu9vTULD4rXgC6QUPFX1nGHThMlKZ8NeMOmU+9mdyQdMrIU2g4ikJmk
DnXxSDfgy96hrepMeTmhroEFKaVsOeu+LYJnZ8FQpzm8s+9V0JFQ7z8k2U3tvzmd
m66iUfeTyNvAqXkuVi/Ju5X5/VyGM0ffetYG+0XcmL9O5lcNsKGIUF/hK8H+XQpP
O3SERvW7c/H5Cbs657L+U8rQGYd7ykXlLxs=
-----END PRIVATE KEY-----

Peer Public Key (bytes): -----BEGIN PUBLIC KEY-----
MIICJDCCARcGCSqGSIb3DQEDATCCAQgCggEBAJHxNKMH9qiUXaGcoUHrMcyePS9F
VVD+2mMefJ4kmKxcigLdIyVLCcnml8EIaS89OzjEF47m/f9arV28oio8xHWgRxWp
Q/9xQl35UlwuFIDzwskOVbDrQjAT6dVP3mChCQADX7T0EYMmRyNfgBuAcc2PciDg
cmG/uQmCqBlkY2s3fSx9cNmujhFbP/8aMnCp+Fl2SKvHokLXLquIm5AQwV/BmVaQ
j1aWo4O8IUpuVXkw7nZ/4aptaDFedDMXuQXT7rPZfm6K1zsLKg7gPHm5n685RBLQ
5qurOl8AEuIpL9UycyfGFKmNTnaB+vKNMWfqNSPZmmXBNQrBu9CpKtYYx88CAQID
ggEFAAKCAQAluvNGz1NYc5b5mQTwnQTjVaY0R5Q0rkEqyCJm387VhhgrMsjE4bIU
BMCOfLSnSyYYxy2UfzlNV+jUUpAzvzbD9ZkubGuFafVp9WQ0ucNeE2rUqFFhHkxA
EHahne/QsD8jMlyLtm/EGS/hDhmHOZTOvcT9v9Dqk2b/DO9tO6SrU+CF0ChpwtUO
mJyGMCQuE5QzLTbnWb5R1MHmxEbtO6IIHzOYCaTrI6E/diVxN0OO82XV2y6bAo/5
UCKo7jywUTOphCL+wcD5nvMbxQ+FaufcsYKSMXbzAJuuOXXRHWQv7UtF3X5VLWH8
vV2F0jUft8VCWaX1/Mi7/tNQ1bCiVmtM
-----END PUBLIC KEY-----

Private Key Numbers: <cryptography.hazmat.primitives.asymmetric.dh.DHPrivateNumbers object at 0x0000025E27E6E830>
Peer Public Key Numbers: <cryptography.hazmat.primitives.asymmetric.dh.DHPublicNumbers object at 0x0000025E27F3FF40>
Error computing shared key: Error computing shared key.
Error computing shared keys: Error computing shared key.
CPU Usage: 8.5%
Memory Usage: 82.0%

C:\Users\world\Documents\ECE 4300>
```

Fig. 1. Results from Python testing

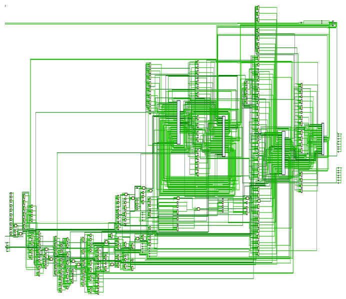
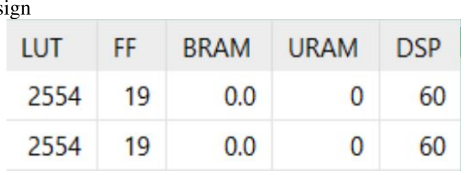
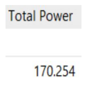


Fig. 2. Design



LUT	FF	BRAM	URAM	DSP
2554	19	0.0	0	60
2554	19	0.0	0	60

Fig. 3. Look up table usage



Total Power
170.254

Fig. 4. Power usage

help accelerate software implementations. We can see that the amount of LUT tables we use is around 2554 with 19 FF in total being used. We are using around 170 watts of power. We can see that compared to the Python version, FPGAs can provide a low memory usage that can be scaled up into larger systems. Of course, there is the ease of use when working on Python implementations compared to FPGAs which can be factored into scaling systems.

IV. Conclusion

Our findings on benchmarking the Diffie-Hellman key exchange provide an interesting opportunity to investigate the importance of resource utilization when implementing a secure network. We have only touched the surface when it comes to implementing benchmarks such as these. In future implementations, the importance of testing FPGA on computers should be looked into and see if that is a more viable option than purely doing hardware or software.