**Cryptography Project – CryptoBench**

**By Stuart Brophy**

**C00159267**

Contents

[1 - Introduction 2](#_Toc152350121)

[Environment Setup: 2](#_Toc152350122)

[Algorithms Used: 2](#_Toc152350123)

[2.1 Plot Graphs 3](#_Toc152350124)

[RSA Keypair Generation: 3](#_Toc152350125)

[RSA Encryption: 4](#_Toc152350126)

[RSA Decryption: 5](#_Toc152350127)

[RSA Digital Signature: 6](#_Toc152350128)

[RSA Verify Signature: 7](#_Toc152350129)

[DSA Keypair Generation: 8](#_Toc152350130)

[DSA Digital Signature: 9](#_Toc152350131)

[DSA Verify Signature: 10](#_Toc152350132)

[EC Keypair Generation: 11](#_Toc152350133)

[EC Digital Signature: 12](#_Toc152350134)

[EC Verify Signature: 13](#_Toc152350135)

[2.1 Results 14](#_Toc152350136)

[(i) Keypair Generation 14](#_Toc152350137)

[(ii) Encryption 14](#_Toc152350138)

[(iii) Decryption 14](#_Toc152350139)

[(iv) Digital Signature 14](#_Toc152350140)

[(v) Verify Signature 14](#_Toc152350141)

[3. Conclusion 15](#_Toc152350142)

[4. Links 16](#_Toc152350143)

# 1 - Introduction

## Environment Setup:

Testing what conducted on two platforms. [CodeSpace](https://github.com/features/codespaces) and [DeepNote](https://deepnote.com).

My CodeSpace environment is 4 CPU cores and 16gb of RAM.

Cores from AMD’s EPYC 7763 Processor

Kali Linux version 2023.3

Python version 3.11.6

Cryptography Library version 38.0.4

My DeepNote environment is 2 CPU cores and 5gb of RAM.

Cores from AMD’s EPYC 7R13 Processor

Debian Linux version 10 (Buster)

Python version 3.9.17

Cryptography Library version 36.0.2

## Algorithms Used:

RSA with key-sizes: 1024, 2048, 3072, 7680, and 15360

DSA with key-sizes: 1024, 2048, and 3072

EC with Algorithms: secp192r1, secp224r1, secp256r1, secp384r1, and secp521r1

# 2.1 Plot Graphs

## RSA Keypair Generation:

CodeSpace:

DeepNote:

## RSA Encryption:

CodeSpace:

DeepNote:

## RSA Decryption:

CodeSpace:

DeepNote:

## RSA Digital Signature:

CodeSpace:

DeepNote:

## RSA Verify Signature:

CodeSpace:

DeepNote:

## DSA Keypair Generation:

CodeSpace:

DeepNote:

## DSA Digital Signature:

CodeSpace:

DeepNote:

## DSA Verify Signature:

CodeSpace:

DeepNote:

## EC Keypair Generation:

CodeSpace:

DeepNote:

## EC Digital Signature:

CodeSpace:

DeepNote:

## EC Verify Signature:

CodeSpace:

DeepNote:

# 2.1 Results

## (i) Keypair Generation

The keypair generation process varies significantly between RSA, DSA, and EC algorithms. As we expected, RSA and DSA times increased with the use of large key-sizes. However, keypair generation for EC algorithms were in general, much faster than RSA or DSA.

## (ii) Encryption

The time it took to encrypt data using RSA was dependent on the key size used, as this dictates the maximum amount of plaintext that can be encrypted. While 1024, 2048 and 3072 all managed to encrypt data at almost the same speed, RSA 7680 and 15360 took significantly longer – It is worth noting however, that this process still took 0.2 seconds or less for even the largest RSA key size.

## (iii) Decryption

The time it took to decrypt data using RSA was again dependent on the key size used. Again, we see RSA 1024, 2048 and 3072 performing well with times <0.001. We see a similar scenario in the case of 7680 and 15360 keys, as the time jumps significantly when decrypting ciphertext of this size.

## (iv) Digital Signature

Digital signing times varied between the different algorithms and key sizes. RSA and DSA signing times increased with key size, with 15360 and 3072 taking the most time, while EC results were for the most part similar to each other. It is worth noting however, that algorithm “secp384r1” was ~x4 longer than others in this process.

## (v) Verify Signature

Signature verification times showed a similar pattern to signing times. RSA and DSA verification times increased with key size, while ECC verification times were faster and showed only slight variation apart from algorithms “secp384r1” and “secp521r1” – The largest available EC algorithms.

# 3. Conclusion

The results I was able to produce were all in line with my expectations when it came to RSA, DSA, and EC algorithms. The operations in relation to RSA and DSA took longer as we increased in key-size, while EC algorithms all seemed to perform as expected, with just a slight increase in time as we use more advanced algorithms/security bits.

An unexpected result was the small difference in times between the different ECC curves. This suggests that even the larger ECC curves can provide increased security with minimal impact on performance.

Another unexpected result was how DeepNote’s environment out-performed CodeSpace is every department, despite having half the CPU cores. This is due to the higher clock-speed per core of DeepNote’s machines, combined with the platform specialising in this type of workflow.

The Python code and resulting data for this project can be found at the following link: [GitHub CrytoBench Repo](https://github.com/StuartBrophy/CrytoBench23)

Overall, my project provided me with valuable insights into how these cryptograph algorithms perform and their characteristics of different cryptographic algorithms and key sizes. These results can help inform the choice of algorithm and key size in different contexts, balancing the need for security with the constraints of computational resources.

# 4. Links

GitHub Repo:

<https://github.com/StuartBrophy/CrytoBench23>

DeepNote Environment:

<https://deepnote.com/workspace/cryptobench23-4a5e0491-eea1-4efd-84d5-466265f337b0/project/CryptoBench23-8bcbdc9e-000a-4d1e-991f-9bd94ba4daaf>

CodeSpace Results:

<https://docs.google.com/spreadsheets/d/e/2PACX-1vTxP-YTPVrPvHIXrK5RtwkVkKPYDhcUeFf2RUy5HpJAmtrg7IeiROWXGk3eftz6z2UynF5H04dT6c9z/pubhtml>

DeepNote Results:

<https://docs.google.com/spreadsheets/d/e/2PACX-1vTK99i_NbzwsxK54iuLhyrC4crRb6BDez9GVjCV61kADEisrLPrOFQkfmWSiXPwtFjXdD3QEV3EOT5v/pubhtml>