Analyzing Programming Language Performance on Simon - a Lightweight Block Cipher

Surya Keswani
Donnie Stewart

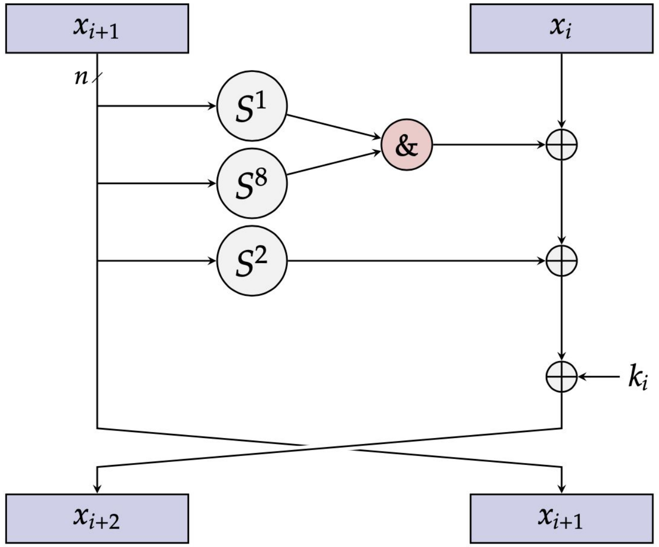
Our Goal: Have a better understanding of how parsers and interpreters affect the performance of a a program

How? Writing the same program in 3 different languages and running a set of performance tests on the program

What Metrics?

- Execution time
- CPU User usage
- CPU System usage
- Memory usage
- Raw compression
- Disk Compression
- Lines of Code

Simon



- ➔ A lightweight block cipher created by the NSA
- ➔ Same security provided as AES 128/256
- ➔ Why did we pick Simon? We are in cryptography and we want to also know how Simon performs with execution time, memory space, etc

How did we Implement Simon?

Input: Hexadecimal Strings

1. Converted inputs to binary bit list
2. Used list to encrypt and decrypt
3. Bit list converted back to hex string
4. Hex strings compared to test vector ↓ ↓ ↓

Plaintext:	`74206e69206d6f6f6d69732061207369`
Ciphertext:	`8d2b5579afc8a3a03bf72a87efe7b868`
Key:	`1f1e1d1c1b1a191817161514131211100f0e0d0c0b0a09080706050403020100`

Simon in python

```
Average Time: 0.0256 seconds

Average Space: 10.4917 MB

Average CPU (User, System): 0.0159 seconds, 0.0010 seconds

Lines of Code: 189

Compression: 6,129 bytes → 2,111 bytes (2.90x) OR 8KB → 4 KB on disk
```

Pro: Easy to switch between types of inputs (hex↔ bit list ↔ decimal ↔ string)

Con: Our First Implementation so VERY HARD to debug crazy long hex numbers and bits

Simon in

Average Time: 1.1820 seconds

Average Space: 3.539 MB

Average CPU (User, System): 0.0017 seconds, 0.0001 seconds

Lines of Code: 276

Compression: 7,553 bytes → 2,695 bytes (2.80x) OR 8KB → 4 KB on disk

Pro: Second implementation so much easier to debug and look at python version for reference

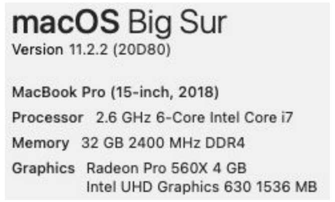
Con: Difficult to switch between all the types we used

Simon in GO

```
Average Time: 0.000048 seconds

Average Space: 2.6037 MB

Average CPU (User, System): .0001938 seconds, 0.0005185 seconds

Lines of Code: 245

Compression: 6,940 bytes → 2,272 bytes (3.05x) OR 8KB → 4 KB on disk
```

Pro: Wins in every performance metric category

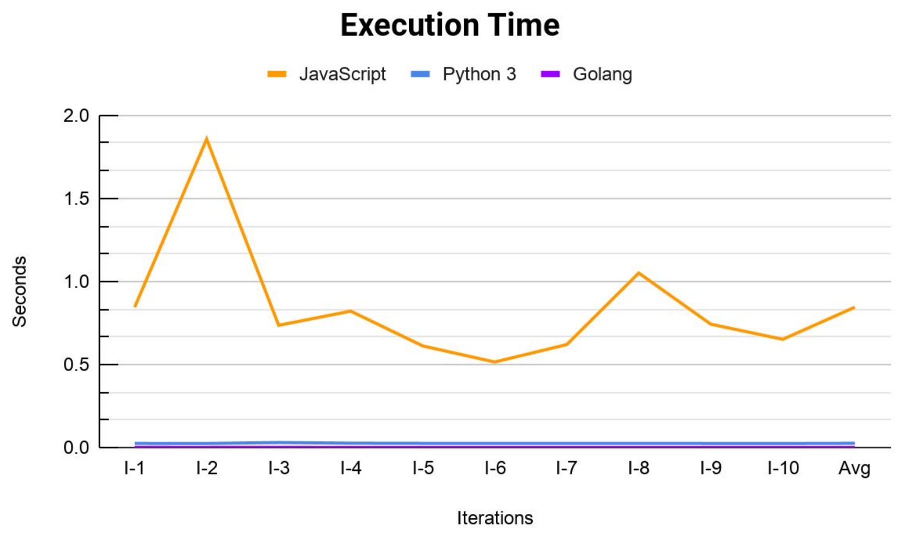
Con: Hard to learn, less resources online compared to js and python3



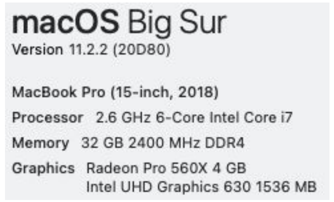
macOS Big Sur
Version 11.2.2 (20D80)

MacBook Pro (15-inch, 2018)
Processor 2.6 GHz 6-Core Intel Core i7
Memory 32 GB 2400 MHz DDR4
Graphics Radeon Pro 560X 4 GB
Intel UHD Graphics 630 1536 MB

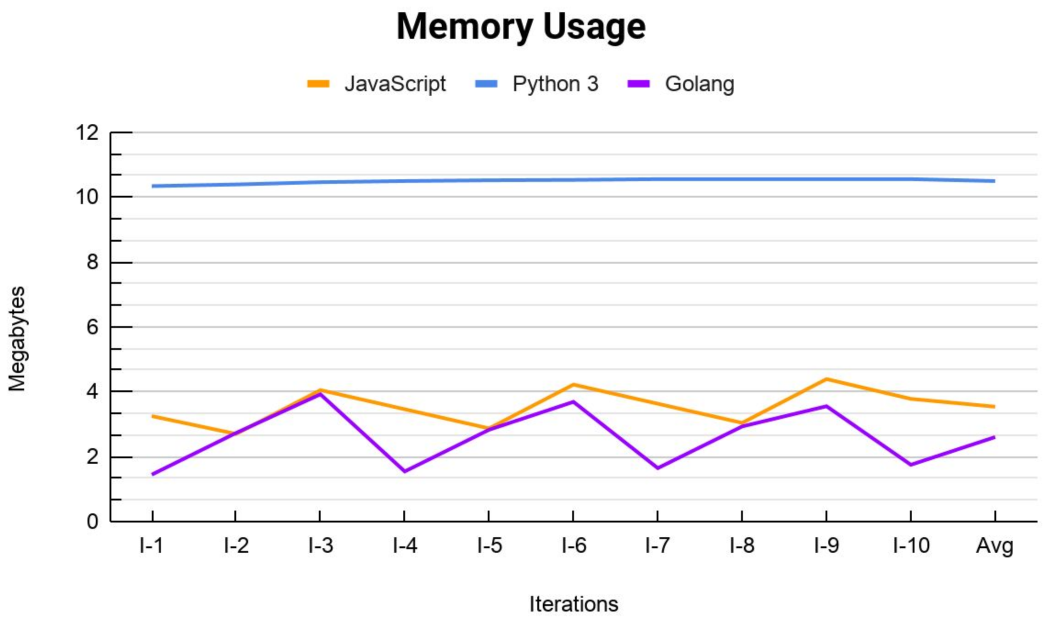
Execution Time



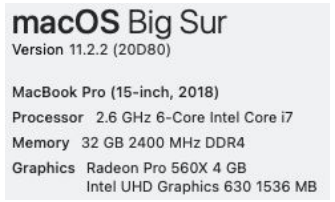
Memory Usage


macOS Big Sur
Version 11.2.2 (20D80)

MacBook Pro (15-inch, 2018)
Processor 2.6 GHz 6-Core Intel Core i7
Memory 32 GB 2400 MHz DDR4
Graphics Radeon Pro 560X 4 GB
Intel UHD Graphics 630 1536 MB

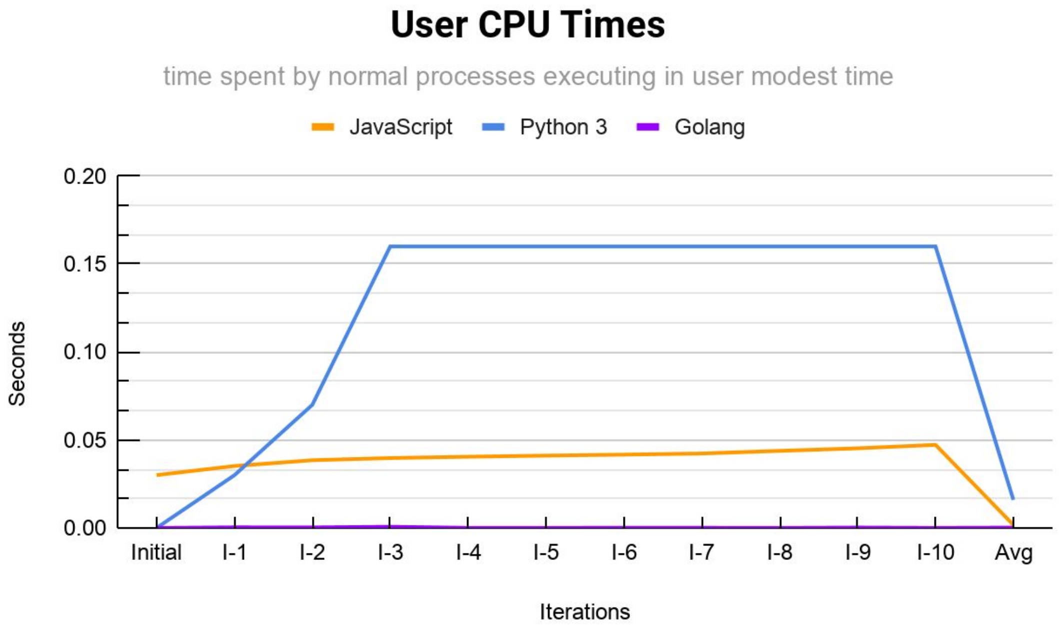


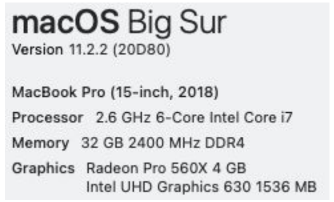
User CPU



macOS Big Sur
Version 11.2.2 (20D80)

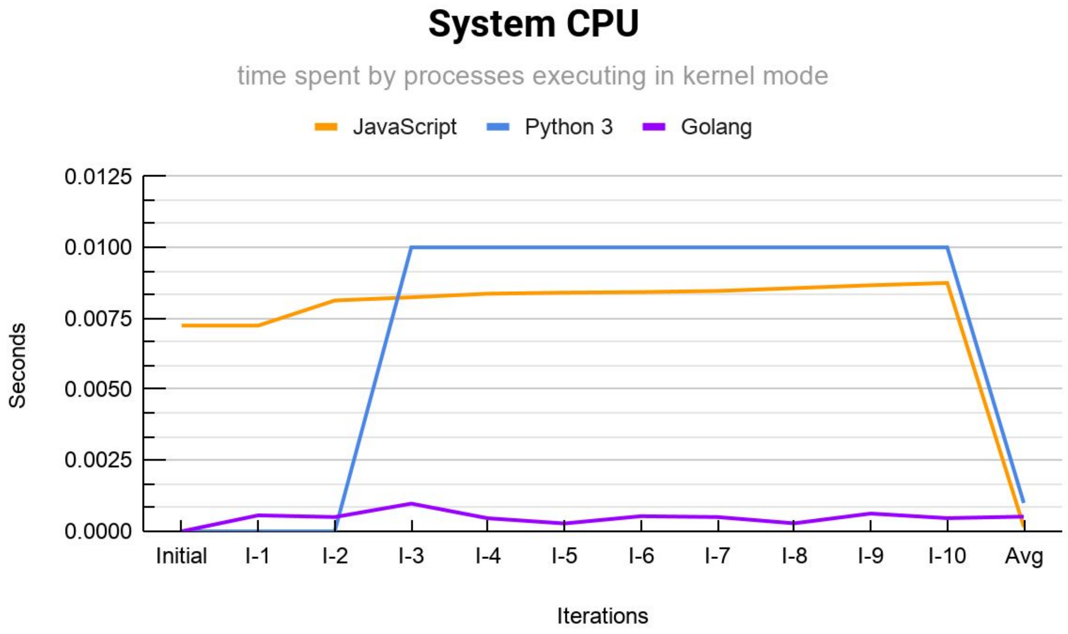
MacBook Pro (15-inch, 2018)
Processor 2.6 GHz 6-Core Intel Core i7
Memory 32 GB 2400 MHz DDR4
Graphics Radeon Pro 560X 4 GB
Intel UHD Graphics 630 1536 MB




macOS Big Sur
Version 11.2.2 (20D80)

MacBook Pro (15-inch, 2018)
Processor 2.6 GHz 6-Core Intel Core i7
Memory 32 GB 2400 MHz DDR4
Graphics Radeon Pro 560X 4 GB
Intel UHD Graphics 630 1536 MB

System CPU



Performance Winners

Execution time: GO

CPU User usage: GO

CPU System usage: GO

Memory usage: GO

Raw compression: GO

Disk Compression: 3 Way Tie

Lines of Code: python

Other Metrics (Future Work)

- ➔ Overall energy usage*
- ➔ Code Readability
- ➔ Split Encryption / Decryption metrics
- ➔ Metrics on other processors and machines
- ➔ Other compression standards

* Website Carbon Calculator

Thanks for Listening.
Questions?