Homomorphic Encryption Implementation for Small SWaP Platforms

2020.03.15

Daniel Achee
Jennifer Quintana

PROJECT INTRODUCTION & BACKGROUND

What is homomorphic encryption?

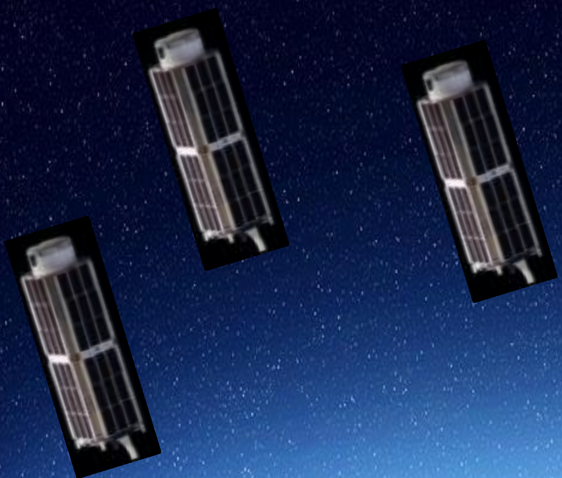
- Homomorphic encryption:
  - An encryption scheme that allows the processing of data in its encrypted form without access to the secret key
  - Preserves the structure of the underlying data
  - Some developed using public asymmetric key systems, such as
    - RSA
    - ElGamal
    - Paillier
- Encryption schemes can be fully homomorphic or partially homomorphic, with respect to one type of data operation (i.e. multiplication, addition)

Types of Homomorphic Encryption

Year	Acronym	Name	Description
2011	BGV	Brakerski-Gentry-Vaikuntanathan	RLWE lattice-based FHE
2011	BFV	Brakerski-Fan-Vercauteren	RLWE lattice-based FHE
2016	CKKS	Cheon-Kim-Kim-Song	Supports approx. arithmetics over complex numbers. Exploits ring isomorphism.

There are many homomorphic encryption schemes, including Enhanced Homomorphic Cryptosystem (EHC), Algebra Homomorphic Encryption (AHEE), and Non-interactive Exponential Homomorphic Encryption Scheme (NEHE); however, we focused most on BFV and CKKS, as they had the most materials available to researchers for free.

Project Motivation:
Space Applications, IoT Devices



- Secure computation for space applications:
  - Cubesats for proprietary research missions with low-SWaP processors can transmit their encrypted data, and securely have data processed and transmitted back, using homomorphic encryption
  - Client-server model

Project Objectives

1. Analyze and implement a series of homomorphic encryption functions
   - Small SWaP Platform:
     - small SoCFPGA
     - FPGA-only
     - ARM processor only
2. Benchmark homomorphic encryption algorithms on cloud computing service
   - Amazon AWS EC2 cloud computing service
3. Implement client-server model
   - Modification of SEAL libraries to split computing requirements for implementation
     - Uses a combination of small SWaP device and a cloud computing service

Homomorphic Encryption Library: Microsoft SEAL (MIT)

- Simple Encrypted Arithmetic Library (SEAL):
  - Open-source software library developed by Microsoft
  - Implements various forms of homomorphic encryption
  - Standard C/C++ (no external dependencies)
  - Supports both asymmetric and symmetric encryption algorithms
  - Supports both BFV and CKKS encryption schemes

Technical Approach

- Implement a low-SWaP client-server model for expanded use of homomorphic encryption schemes
  - Some existing experiments for IoT devices and high-performance computing platforms
    - Limited implementations achievable with Raspberry Pi in literature
  - No current literature on low-SWaP space applications
    - Desire for better encryption for cubesat and small sats, resistance to quantum computing for forced decryption
  - No library of functions for FPGA implementation commercially available
    - Some in-work at IBM, pay for access

Two CubeSats, part of a constellation built and operated by Planet Labs Inc. to take images of Earth, were launched from the International Space Station on May 17, 2016.
(Image: © NASA)

- Relevant current solicitation for research FY2020 – FY2023:
  - AFRL BAA *Capabilities for Cyber Resiliency*
  - AFRL BAA *Foundations of Trusted Computational Information Systems*
  - NAVAIR BAA *Cyber Warfare Detachment*
  - AFRL BAA *Next Generation Intelligence Collection and Analysis*
  - AFRL BAA *Measurement and Signatures Intelligence Exploitation*

Related Work

- Simulated WBAN through OMNET++ and Castalia
  - 30s interval for reading and writing encrypted data
  - Simulation time 3600 s
  - Simulation with Helib and SEAL
    - Measured packet delay

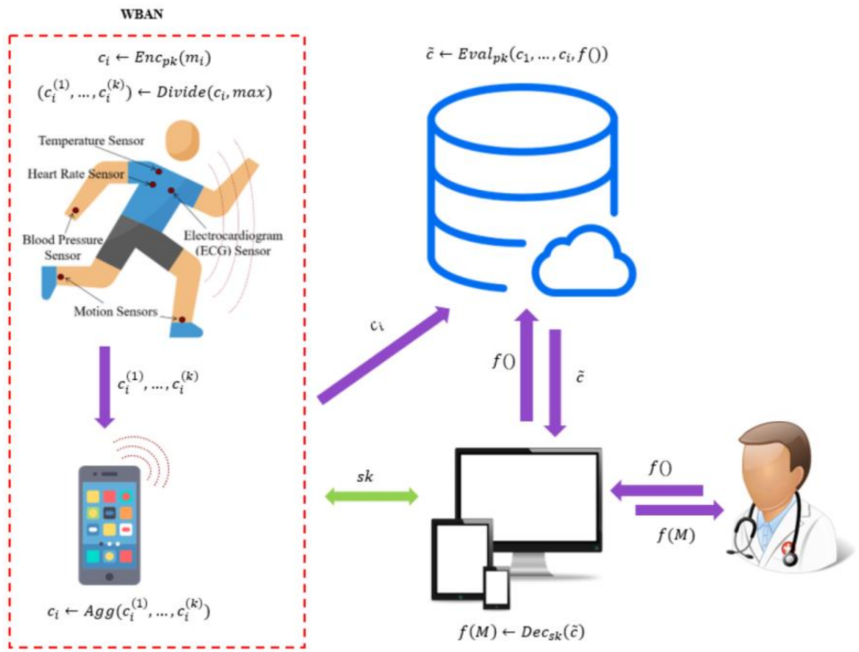


Table 5. The running time (Milliseconds)

		KeyGen	Encryption	Decryption	Addition	Multiplication	Bootstrapping
PC	HElib	0.760058	0.032094	0.013368	0.000094	0.066178	85.323830
	SEAL	1.930100	2.342723	0.177101	0.005329	2.187750	-
Raspberry Pi	HElib	79.933075	2.084733	1.258043	0.006370	4.707492	7,846.207000
	SEAL	181.319900	229.979548	46.673325	0.920642	480.622600	-

A. Prasitupparote, et. Al., “Implementation and Analysis of Fully Homomorphic Encryption in Wearable Devices”, *Proceedings of 4th International Conference on Information Security and Digital Forensics, ISDF2018,* Greece, 2018.

Objective 1

Objective 1: Small SWaP Platform Analysis

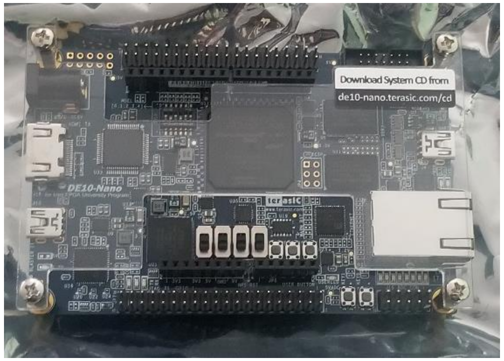
Item No.	Manufacturer	Device Description	Platform Type
1	Intel	DE10 Nano SoC (Cyclone V)	SoC (ARM + FPGA)
2	Intel	DE0 Nano (Cyclone IV)	FPGA
3	Xilinx	Arty A7 (Zynq 7020)	SoC (ARM + FPGA)
4	Texas Instruments	MSP430 LaunchPad	Microcontroller
5	Raspberry Pi	Rpi 4	ARM Processor

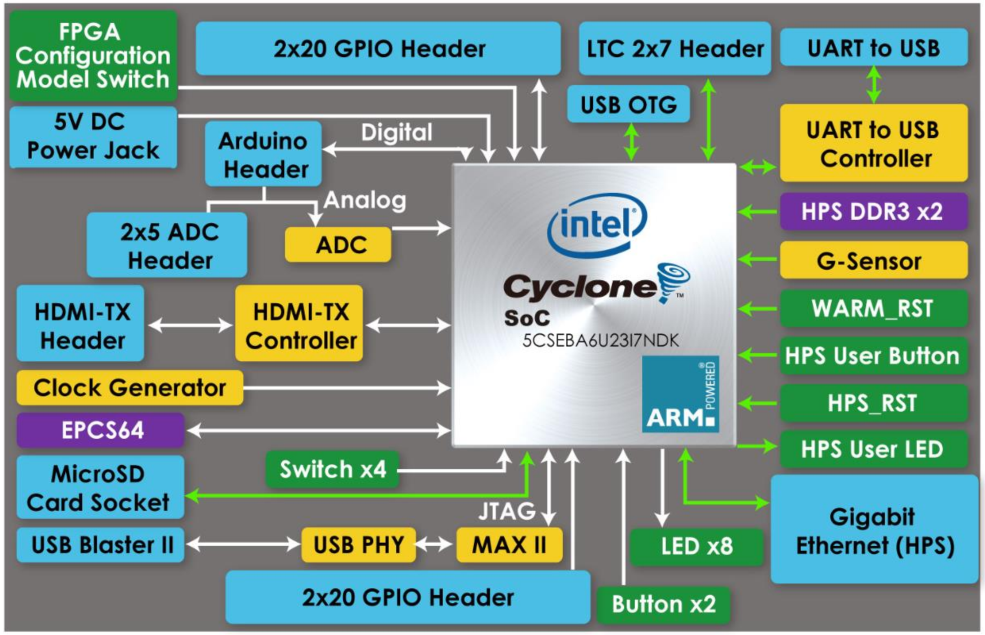
Criteria examined:

- Power consumption
- Processor type
- FPGA number of programmable logic elements
- Physical size
- Cost

Objective 1:
Small SWaP Platform Selection 1

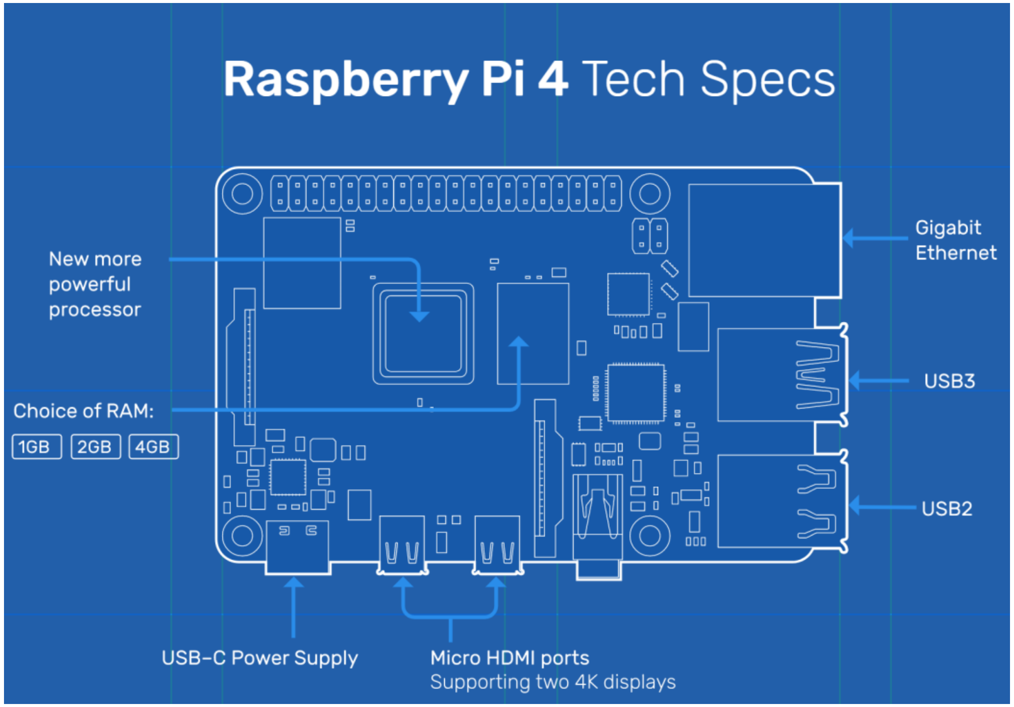
- Intel Cyclone V:
  DE10 Nano SoC
  - SoCFGPA
    - Dual-core Cortex A9
      32-bit ARM processor
    - 110k Programmable
      Logic Elements
- Linux OS:
  - Yocto Poky Distro
  - Boot from 32GB SD
    card





Objective 1:
Small SWaP Platform Selection 2

- Raspberry Pi 4
  - Cortex A72 quad-core 64-bit ARM processor
  - 4GB DDR3 memory



Objective 1: Implementation

- Using both DE10 Nano SoC and Raspberry Pi 4, we attempted to compile the SEAL library functions and run on the platform.
  - Using DE10 Nano SoC ARM processor
  - Porting C -> HDL on DE10 Nano SoC FPGA
  - Using Raspberry Pi 4 ARM processor

Objective 1: Results

- 32-bit processor incompatible with SEAL and similar available homomorphic encryption libraries.
- Requires at least 64-bit processor.
- HDL -> C implementation too large for Cyclone V FPGA
- Success with Raspberry Pi 4 Cortex A78 ARM processor
  - All instances runnable for degrees <32768

Objective 1: Results (2)

#	Parameter	SEAL BFV Degree Computation Time [ms]				
	Degree	1024	2048	4096	8192	16384
1	Average Batch	0.233	0.466	0.984	2.092	4.435
2	Average Unbatch	0.186	0.344	0.740	1.569	3.373
3	Average Encrypt	3.329	6.524	22.071	64.670	219.878
4	Average Decrypt	1.566	2.417	7.137	26.089	100.465
5	Average Add	0.025	0.036	0.116	0.498	2.047
6	Average Multiply	11.460	22.102	74.159	279.802	1,146.392
7	Average Multiply Plain	1.006	2.076	8.810	37.527	158.410
8	Average Square	8.252	16.237	55.181	209.815	867.853
9	Average Relinearize	N/A	N/A	16.476	72.648	396.322
10	Average Rotate 1 step	N/A	N/A	16.613	73.975	400.704
11	Average Rotate Random	N/A	N/A	53.823	341.561	1,765.455
12	Average Rotate Columns	N/A	N/A	16.624	73.992	400.718
13	CPU Usage [%]	5.0 %	10.6%	87.7%	99.7%	100%

Objective 1: Results (3)

#	Parameter	SEAL CKKS Degree Computation Time [ms]				
	Degree	1024	2048	4096	8192	16384
1	Average Batch	1.801	2.969	7.514	20.257	61.416
2	Average Unbatch	2.594	4.250	14.735	57.825	257.827
3	Average Encrypt	3.016	4.455	21.417	66.505	232.468
4	Average Decrypt	0.148	0.215	0.842	3.325	13.085
5	Average Add	0.027	0.036	0.114	0.446	1.998
6	Average Multiply	0.427	0.729	2.951	12.627	48.889
7	Average Multiply Plain	0.186	0.316	1.243	4.921	19.506
8	Average Square	0.302	0.517	2.137	9.173	36.203
9	Average Relinearize	N/A	N/A	16.323	71.417	391.350
	Average Rescale	N/A	N/A	6.235	30.999	140.706
10	Average Rotate 1 step	N/A	N/A	16.734	73.160	399.765
11	Average Rotate Random	N/A	N/A	57.636	279.600	1,790.253
12	Average Complex Conjugate	N/A	N/A	16.684	73.036	398.881
13	CPU [%]	7.0%	11.6%	81.1%	98.3%	100%

OBJECTIVE 2

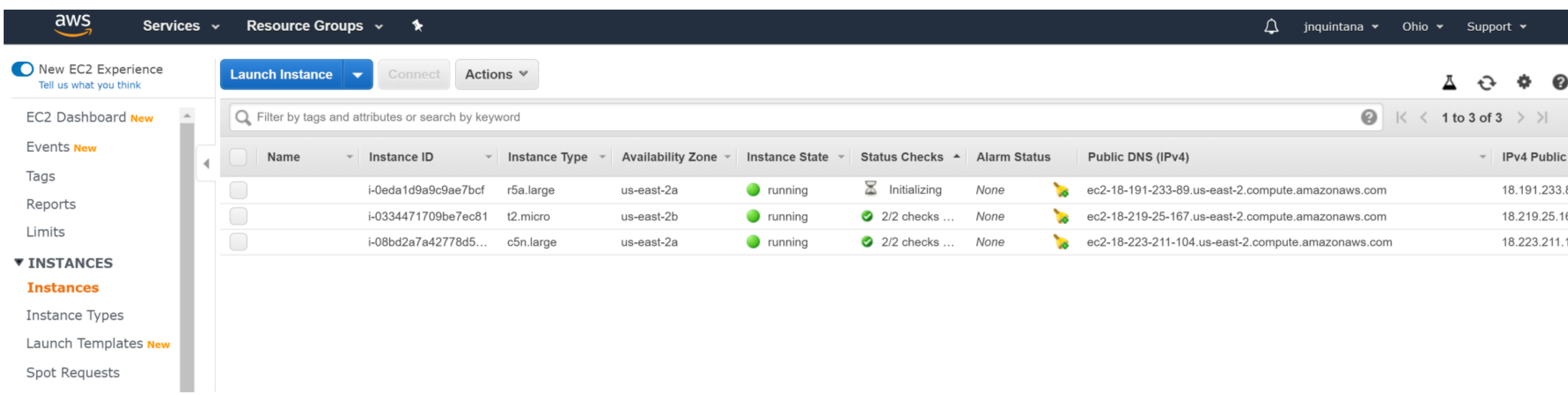
Objective 2: AWS EC2 Selection

Item No.	Instance Name	Instance Type	CPU Cores	Memory
1	t1.micro	General Purpose	1	0.5 GB
2	c5n.large	Computation Optimized	2	4 GB
3	t2.micro	General Purpose	1	1 GB
4	r5a.large	Memory Optimized	2	16 GB

- Selected different instances based on processing properties:
  - Large, high-capability instances
  - Small, resource-limited instances

Objective 2: Implementation (1)

- Instantiated AWS EC2:
- Ubuntu 18.04 OS used for instantiations
- SSH to EC2 to run performance metrics for different functional implementations of SEAL benchmarking



Objective 2: Results for t2_micro instance - BFV

#	Parameter	SEAL BFV Degree Computation Time [ms]				
Degree		1024	2048	4096	8192	16384
1	Average Batch	0.024	0.050	0.129	0.273	0.539
2	Average Unbatch	0.026	0.048	0.105	0.214	0.453
3	Average Encrypt	0.537	1.034	3.034	8.262	26.371
4	Average Decrypt	0.118	0.234	0.764	2.671	11.488
5	Average Add	0.005	0.008	0.034	0.112	0.430
6	Average Multiply	1.103	2.180	7.349	27.757	116.834
7	Average Multiply Plain	0.113	0.234	0.931	4.047	19.624
8	Average Square	0.754	1.509	5.038	20.058	82.345
9	Average Relinearize	N/A	N/A	1.496	7.331	42.251
10	Average Rotate 1 step	N/A	N/A	1.549	7.354	43.365
11	Average Rotate Random	N/A	N/A	6.213	31.027	205.262
12	Average Rotate Columns	N/A	N/A	1.493	7.360	42.124
13	CPU Usage [%]	1%	2.3%	16.6%	51.8%	89.7%

Objective 2:
Results for t2_micro instance - CKKS

#	Parameter	SEAL CKKS Degree Computation Time [ms]				
	Degree	1024	2048	4096	8192	16384
1	Average Batch	0.496	1.055	2.467	6.197	17.045
2	Average Unbatch	0.558	1.182	3.273	11.141	46.447
3	Average Encrypt	0.508	0.950	3.376	9.462	31.022
4	Average Decrypt	0.015	0.026	0.112	0.405	1.705
5	Average Add	0.006	0.009	0.030	0.105	0.418
6	Average Multiply	0.053	0.088	0.345	1.364	5.432
7	Average Multiply Plain	0.014	0.023	0.091	0.375	1.453
8	Average Square	0.035	0.064	0.251	1.015	4.191
9	Average Relinearize	N/A	N/A	1.530	7.811	43.702
	Average Rescale	N/A	N/A	0.733	3.516	15.965
10	Average Rotate 1 step	N/A	N/A	1.750	8.344	47.054
11	Average Rotate Random	N/A	N/A	7.205	31.284	227.630
12	Average Complex Conjugate	N/A	N/A	1.757	8.244	46.406
13	CPU [%]	1.0%	1.7%	15.0%	46.2%	100%

Objective 2: Results for c5n_large instance - BFV

#	Parameter	SEAL BFV Degree Computation Time [ms]				
	Degree	1024	2048	4096	8192	16384
1	Average Batch	0.025	0.054	0.105	0.213	0.425
2	Average Unbatch	0.023	0.039	0.083	0.168	0.352
3	Average Encrypt	0.421	0.786	2.274	6.227	19.964
4	Average Decrypt	0.082	0.165	0.541	1.920	7.527
5	Average Add	0.004	0.006	0.018	0.073	0.309
6	Average Multiply	0.747	1.571	5.255	20.001	81.804
7	Average Multiply Plain	0.081	0.169	0.712	2.999	12.707
8	Average Square	0.522	1.071	3.612	13.791	57.450
9	Average Relinearize	N/A	N/A	1.078	5.273	31.282
10	Average Rotate 1 step	N/A	N/A	1.083	5.310	31.056
11	Average Rotate Random	N/A	N/A	4.467	20.178	136.762
12	Average Rotate Columns	N/A	N/A	1.082	5.291	30.989
13	CPU Usage [%]	1.0%	1.7%	12.0%	36.7%	100%

Objective 2:
Results for c5n_large instance - CKKS

#	Parameter	SEAL CKKS Degree Computation Time [ms]				
	Degree	1024	2048	4096	8192	16384
1	Average Batch	0.364	0.788	1.802	4.487	12.292
2	Average Unbatch	0.390	0.831	2.325	8.039	33.520
3	Average Encrypt	0.422	0.792	2.643	7.334	23.595
4	Average Decrypt	0.009	0.016	0.063	0.245	1.041
5	Average Add	0.004	0.006	0.017	0.068	0.284
6	Average Multiply	0.024	0.048	0.179	0.841	3.900
7	Average Multiply Plain	0.009	0.016	0.060	0.239	0.950
8	Average Square	0.016	0.032	0.120	0.552	2.763
9	Average Relinearize	N/A	N/A	1.097	5.357	31.338
	Average Rescale	N/A	N/A	0.519	2.574	11.756
10	Average Rotate 1 step	N/A	N/A	1.280	6.155	34.147
11	Average Rotate Random	N/A	N/A	4.458	28.515	145.624
12	Average Complex Conjugate	N/A	N/A	1.275	6.045	33.749
13	CPU [%]	0.7%	1.3%	11.0%	41.3%	100%

Objective 2:
Results for r5a_large instance - BFV

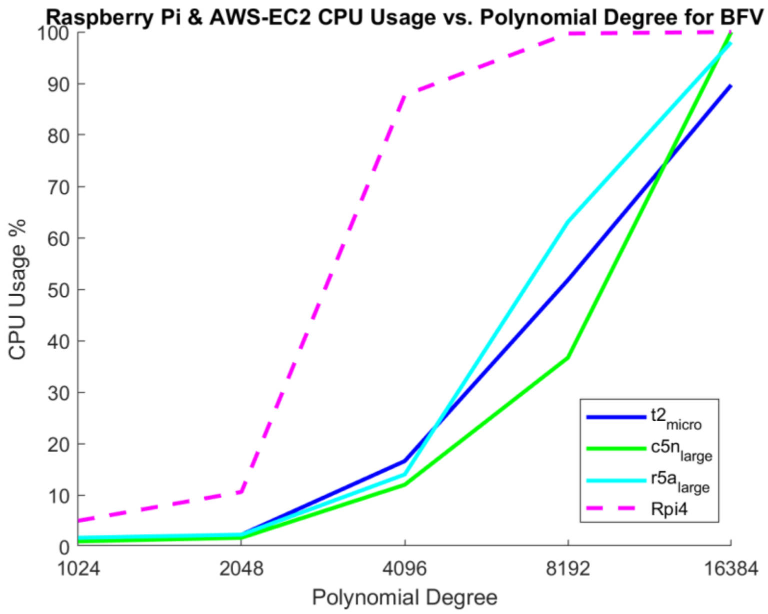
#	Parameter	SEAL BFV Degree Computation Time [ms]				
	Degree	1024	2048	4096	8192	16384
1	Average Batch	0.056	0.056	0.124	0.273	0.513
2	Average Unbatch	0.022	0.046	0.091	0.196	0.451
3	Average Encrypt	0.482	0.902	2.604	7.491	24.381
4	Average Decrypt	0.102	0.186	0.632	2.374	9.425
5	Average Add	0.005	0.007	0.020	0.083	0.327
6	Average Multiply	1.002	1.986	6.581	25.472	103.115
7	Average Multiply Plain	0.102	0.206	0.930	3.827	16.475
8	Average Square	0.676	1.390	4.609	18.096	74.667
9	Average Relinearize	N/A	N/A	1.270	6.631	39.109
10	Average Rotate 1 step	N/A	N/A	1.322	6.585	39.271
11	Average Rotate Random	N/A	N/A	4.299	24.564	162.075
12	Average Rotate Columns	N/A	N/A	1.287	6.708	39.196
13	CPU Usage [%]	1.7%	2.3%	14.0%	63.1%	98%

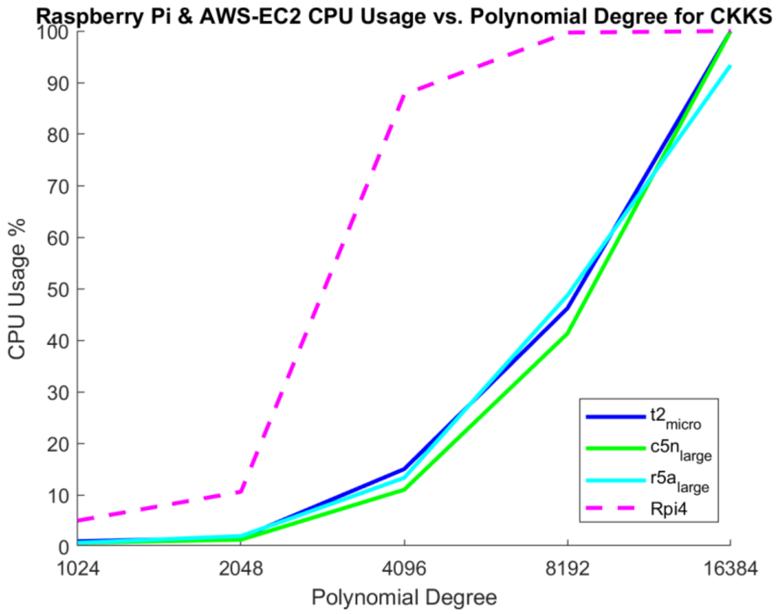
Objective 2:
Results for r5a_large instance - CKKS

#	Parameter	SEAL CKKS Degree Computation Time [ms]				
Degree		1024	2048	4096	8192	16384
1	Average Batch	0.479	1.080	2.227	5.365	14.436
2	Average Unbatch	0.536	1.102	3250	10.403	44.064
3	Average Encrypt	0.481	0.855	3.145	8.735	28.172
4	Average Decrypt	0.012	0.022	0.091	0.302	1.229
5	Average Add	0.004	0.007	0.022	0.075	0.311
6	Average Multiply	0.033	0.066	0.268	0.984	4.578
7	Average Multiply Plain	0.013	0.024	0.100	0.351	1.500
8	Average Square	0.022	0.045	0.179	0.677	3.103
9	Average Relinearize	N/A	N/A	1.449	6.599	38.316
	Average Rescale	N/A	N/A	0.667	3.164	14.578
10	Average Rotate 1 step	N/A	N/A	1.623	7.275	41.515
11	Average Rotate Random	N/A	N/A	5.934	28.260	181.675
12	Average Complex Conjugate	N/A	N/A	1.579	7.215	41.180
13	CPU [%]	0.7%	2%	13.3%	48.7%	93.4%

Objective 2 Results: CPU Usage

UCLA | EE209AS Embedded Systems Cybersecurity



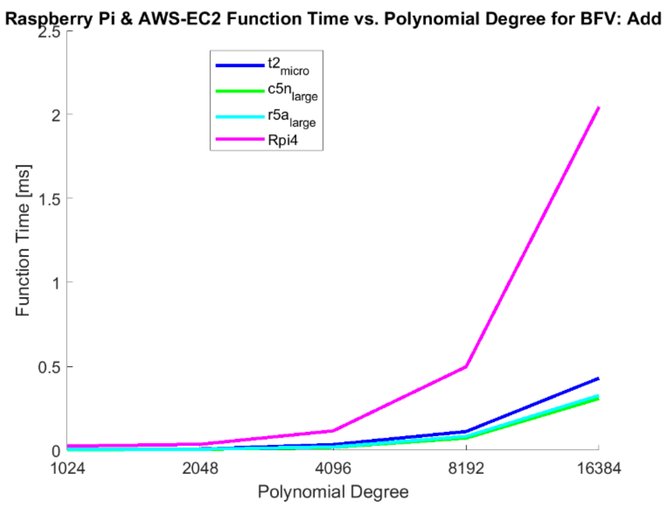


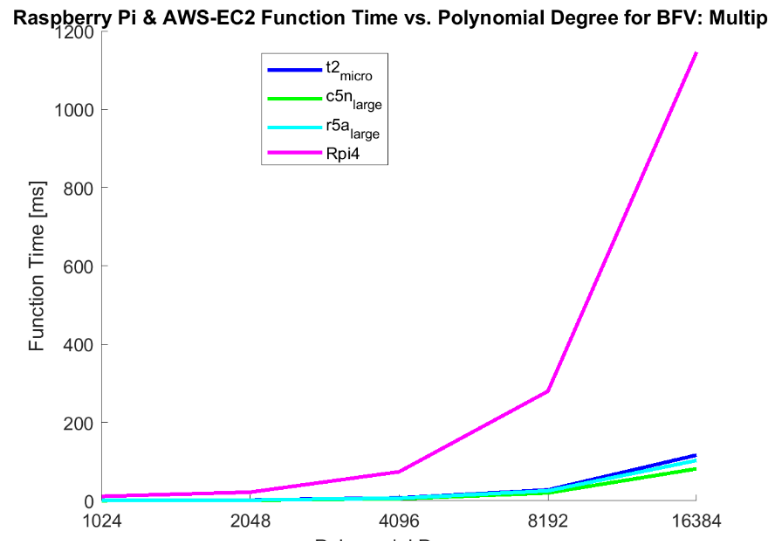
Raspberry Pi cannot complete 32k polynomial degree. AWS EC2 instances can complete without crashing but with long delay times (10s +) for each run.

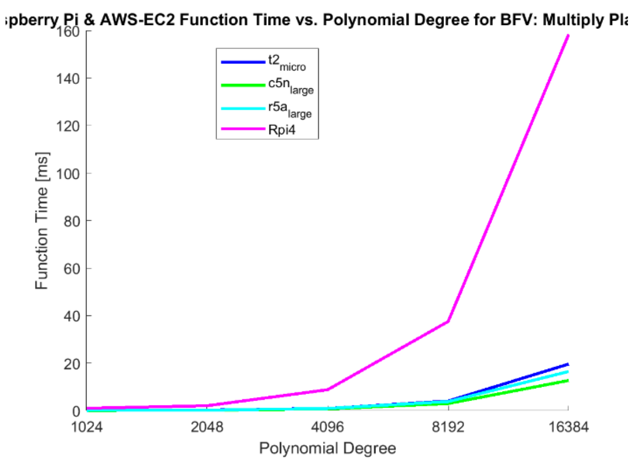
CPU utilization approaches 100% for all current platforms, including computationally optimized and memory optimized AWS EC2 instances.

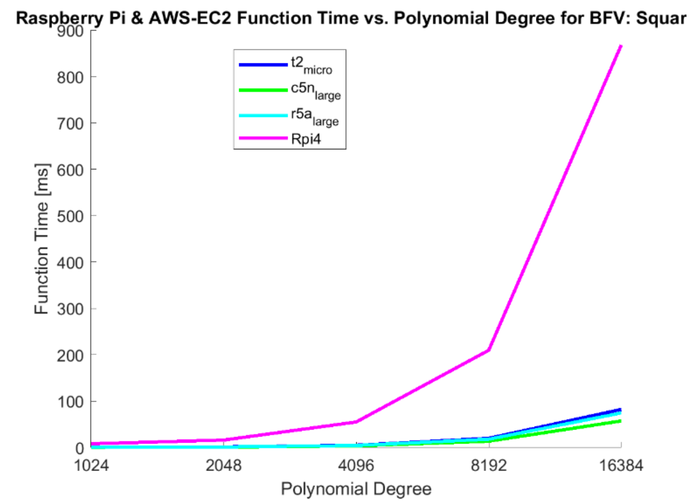
Objective 2 Results: Function Computation Time

UCLA | EE209AS Embedded Systems Cybersecurity









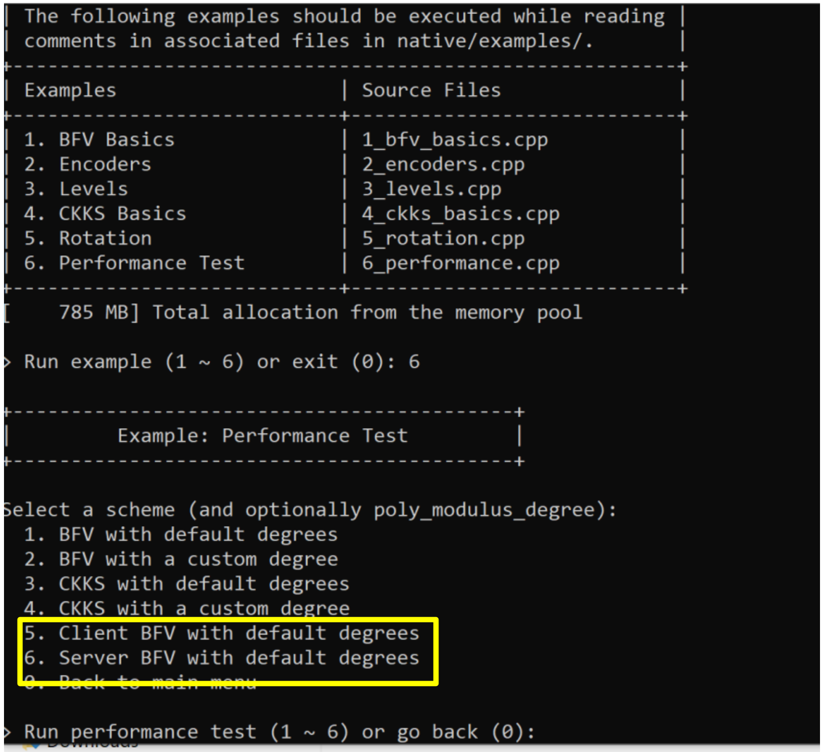
OBJECTIVE 3

Objective 3: Client-Server Model

- Client-server model will move the bulk of the calculations from the resource-constrained Raspberry Pi to the AWS EC2 server and from micro AWS EC2 client and high performance AWS EC2 server.
  - Intended to lessen the computational cost on each device to allow a homomorphic solution for small SWaP devices

Objective 3: Implementation

- Client BFV with 4096 default degree test and server BFV with 4096 default degree test added to SEAL library functions



```
| The following examples should be executed while reading |
| comments in associated files in native/examples/.        |
+----------------------------+----------------------------+
| Examples                   | Source Files               |
+----------------------------+----------------------------+
| 1. BFV Basics              | 1_bfv_basics.cpp           |
| 2. Encoders                | 2_encoders.cpp             |
| 3. Levels                  | 3_levels.cpp               |
| 4. CKKS Basics             | 4_ckks_basics.cpp          |
| 5. Rotation                | 5_rotation.cpp             |
| 6. Performance Test        | 6_performance.cpp          |
+----------------------------+----------------------------+
[   785 MB] Total allocation from the memory pool

> Run example (1 ~ 6) or exit (0): 6

+---------------------------------------------+
|         Example: Performance Test           |
+---------------------------------------------+

Select a scheme (and optionally poly_modulus_degree):
  1. BFV with default degrees
  2. BFV with a custom degree
  3. CKKS with default degrees
  4. CKKS with a custom degree
  5. Client BFV with default degrees
  6. Server BFV with default degrees
  0. Back to main menu

> Run performance test (1 ~ 6) or go back (0):
```

Objective 3: Client-Server Implementation


```
+--------------------------------------------------------------------------------------+
|         Server BFV Performance Test with Degrees: 4096, 8192, 16384, and 32768       |
+--------------------------------------------------------------------------------------+
/
| Encryption parameters :
|   scheme: BFV
|   poly_modulus_degree: 4096
|   coeff_modulus size: 109 (36 + 36 + 37) bits
|   plain_modulus: 786433
\

Running tests .......... Done

Average add: 18157 microseconds
Average multiply: 61216 microseconds
Average multiply plain: 35296 microseconds
Average square: 42591 microseconds

/
| Encryption parameters :
|   scheme: BFV
|   poly_modulus_degree: 8192
|   coeff_modulus size: 218 (43 + 43 + 44 + 44 + 44) bits
|   plain_modulus: 786433
\

Running tests .......... Done

Average add: 65073 microseconds
Average multiply: 228081 microseconds
Average multiply plain: 127401 microseconds
Average square: 162559 microseconds

/
| Encryption parameters :
|   scheme: BFV
|   poly_modulus_degree: 16384
|   coeff_modulus size: 438 (48 + 48 + 48 + 49 + 49 + 49 + 49 + 49 + 49) bits
|   plain_modulus: 786433
\

Running tests .......... Done

Average add: 270092 microseconds
Average multiply: 961099 microseconds
Average multiply plain: 529304 microseconds
Average square: 683535 microseconds
```

```
+--------------------------------------------------------------------------------------+
|         Client BFV Performance Test with Degrees: 4096, 8192, 16384, and 32768       |
+--------------------------------------------------------------------------------------+
/
| Encryption parameters :
|   scheme: BFV
|   poly_modulus_degree: 4096
|   coeff_modulus size: 109 (36 + 36 + 37) bits
|   plain_modulus: 786433
\

Generating secret/public keys: Done
Running tests      + Plaintext polynomial: 2
.    + Plaintext polynomial: 2x^1 + 2
.    + Plaintext polynomial: 4x^1 + 2
.    + Plaintext polynomial: 2x^2 + 2x^1 + 2
.    + Plaintext polynomial: 4x^2 + 2
.    + Plaintext polynomial: 4x^2 + 2x^1 + 2
.    + Plaintext polynomial: 4x^2 + 4x^1 + 2
.    + Plaintext polynomial: 2x^3 + 2x^2 + 2x^1 + 2
.    + Plaintext polynomial: 4x^3 + 2
.    + Plaintext polynomial: 4x^3 + 2x^1 + 2
. Done

Average add: 15929 microseconds
Average multiply: 61121 microseconds
Average multiply plain: 34737 microseconds
Average square: 43107 microseconds

/
| Encryption parameters :
|   scheme: BFV
|   poly_modulus_degree: 8192
|   coeff_modulus size: 218 (43 + 43 + 44 + 44 + 44) bits
|   plain_modulus: 786433
\

Generating secret/public keys: Done
Running tests      + Plaintext polynomial: 2
.    + Plaintext polynomial: 2x^1 + 2
.    + Plaintext polynomial: 4x^1 + 2
.    + Plaintext polynomial: 2x^2 + 2x^1 + 2
.    + Plaintext polynomial: 4x^2 + 2
.    + Plaintext polynomial: 4x^2 + 2x^1 + 2
.    + Plaintext polynomial: 4x^2 + 4x^1 + 2
.    + Plaintext polynomial: 2x^3 + 2x^2 + 2x^1 + 2
.    + Plaintext polynomial: 4x^3 + 2
.    + Plaintext polynomial: 4x^3 + 2x^1 + 2
. Done

Average add: 59338 microseconds
Average multiply: 228033 microseconds
Average multiply plain: 125155 microseconds
Average square: 164971 microseconds
```

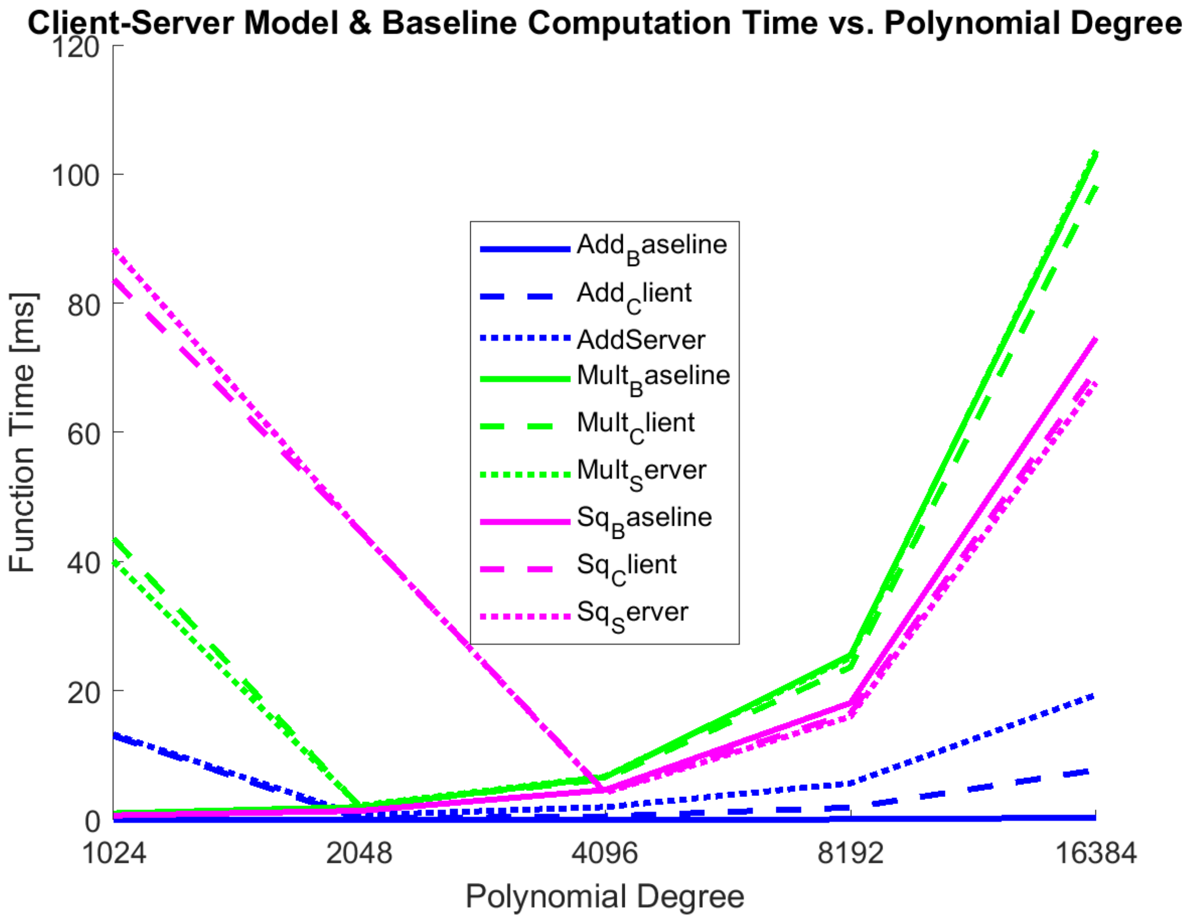
Objective 3: Client Server Results Table (c5n shown)

UCLA | EE209AS Embedded Systems Cybersecurity

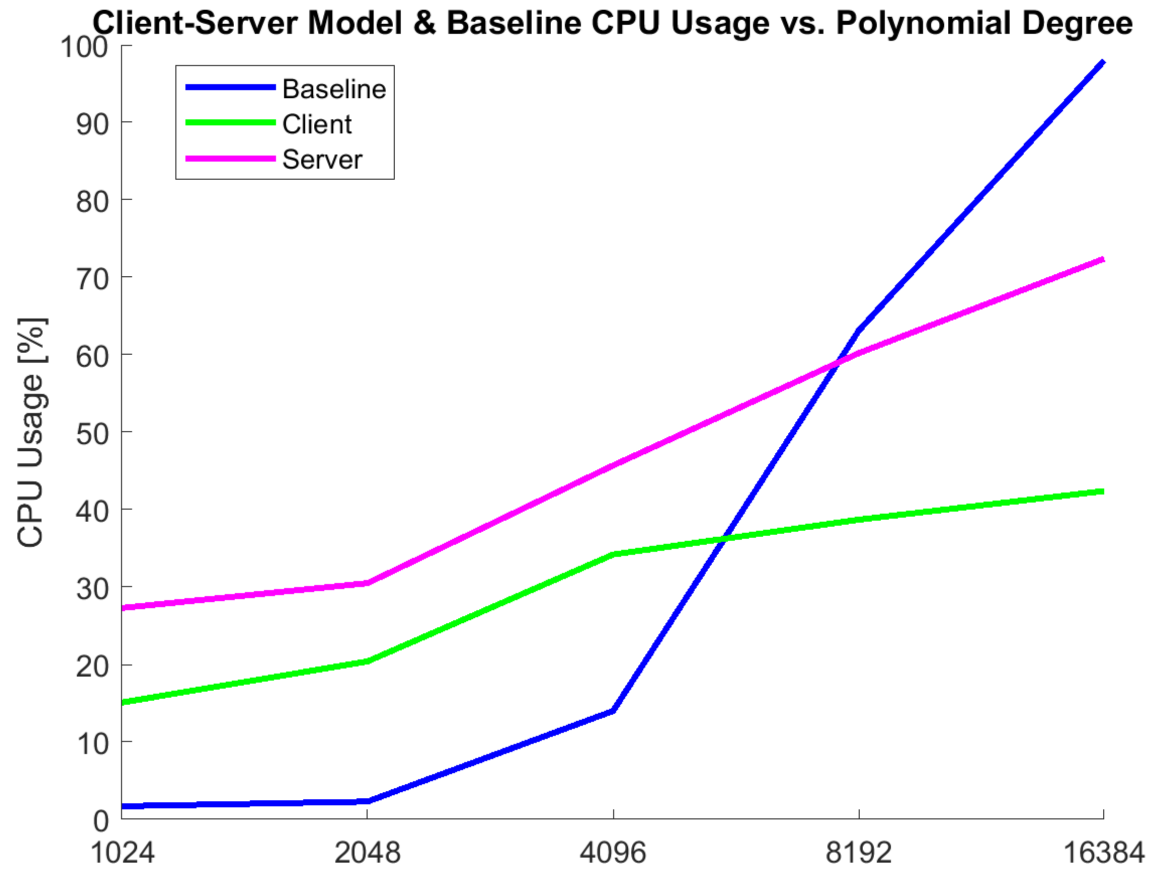
#	Parameter	SEAL BFV Degree (Client) Computation Time [ms]				
	Degree	1024	2048	4096	8192	16384
5	Average Add	13.002	0.226	0.539	1.895	7.71
6	Average Multiply	43.559	2.106	6.294	23.639	98.195
7	Average Multiply Plain	87.966	42.817	18.324	5.619	23.462
8	Average Square	83.679	44.901	4.321	16.539	69.679
13	CPU Usage [%]	15.1%	20.4%	34.2%	38.7%	42.4%

#	Parameter	SEAL BFV Degree (Server) Computation Time [ms]				
	Degree	1024	2048	4096	8192	16384
5	Average Add	13.245	0.712	1.971	5.667	19.364
6	Average Multiply	40.085	2.221	6.708	25.129	103.67
7	Average Multiply Plain	86.588	42.386	18.429	6.14	25.528
8	Average Square	88.393	44.865	4.19	15.996	67.72
13	CPU Usage [%]	27.3%	30.5%	45.7%	60.2%	72.4%

Objective 3: Results – Function Time vs. Baseline



Objective 3: Results – CPU Usage



SUMMARY

Future Work

- Optimization of the SEAL library functions for to improve resource freeing and reallocation
- Improvement of Raspberry Pi platform resource allocation
- Implementation on IoT device app for in-situ encrypted data usage

Summary

- Homomorphic encryption has made advancements in the past decade for implementation.
  - Understood from mathematical perspective
  - Can now optimize for resource-constrained environments
- Client-server model prototype can allow for future IoT device platforms to make use of homomorphic encryption and optimize for practical use.

GitHub File Structure

- README.txt
- 209AS_FinalPresentation.pptx
- 209AS_FinalReport.docx
- Client-Server Model <software folder>
- Console data logs <text file data folder>

- https://github.com/daniel-achee/ece209-project.git
- https://daniel-achee.github.io/ece209-project/

Division of Labor

- Presentation & Report:
  - Primary: J. Quintana
  - Supporting: D. Achee
- Objective 1:
  - Primary: J. Quintana
  - Supporting: D. Achee
- Objective 2:
  - Primary: D. Achee
  - Supporting: J. Quintana
  - Initial AWS-EC2 instance implementation done by D. Achee, data shown from J. Quintana AWS-EC2 instances.
- Objective 3:
  - Server side: D. Achee
  - Client side: J. Quintana
  - Client-server model done by D. Achee, Rpi platform side done by J. Quintana

References

- [1] P.V. Parmar, et. Al., “Survey of Various Homomorphic Encryption Algorithms and Schemes”, *International Journal of Computer Applications*, (0975-8887), Vol. 81, No. 8, April 2014.
- [2] C. Gentry, “Fully Homomorphic Encryption Using Ideal Lattices”, *Stanford University, IBM Watson*, 2009.
- [3] K. Lauter, et. Al., “Can Homomorphic Encryption be Practical?” *Microsoft Research*, *Eindhoven University of Technology, University of Toronto*, 2011.
- [4] N. Downlin, et. Al., “Manual for Using Homomorphic Encryption for Bioinformatics”, *Microsoft Research, Princeton University*, accessed Jan 2020.
- [5] Z. Brakerski and V. Vaikuntanathan, “Efficient Fully Homomorphic Encryption from (Standard) LWE”, *SIAM Journal on Computing*, Vol. 43, No. 2, 831-871, 2014.
- [6] J. Howe, et. Al., “On Practical Discrete Gaussian Samplers for Lattice-Based Cryptography”, *IEEE Transactions on Computers*, Vol. 67, No. 3, March 2018.
- [7] S. Fau, et. Al., “Towards practical program execution over fully homomorphic encryption schemes”, *Eighth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*, 2013.
- [8] K. Laine, et. Al., “Simple Encrypted Arithmetic Library 2.3.1”, *Microsoft Research*, 2018.
- [9] A. Prasitsupparote, et. Al., “Implementation and Analysis of Fully Homomorphic Encryption in Wearable Devices”, *Proceedings of 4th International Conference on Information Security and Digital Forensics, ISDF2018*, Greece, 2018.