Studying the Implementations and Performance of the Lightwight Cryptographic TinyJAMBU Algorithm Between C++ Compilers

Rob Ranit
Electrical and Computer Engineering Department
California State Polytechnic University
Pomona, CA, US
rranit@protonmail.com

***Abstract*—This research paper focuses on the performance of a C++ implementation of the lightweight cryptographic algorithm TinyJAMBU, focusing specifically on the differences of compiling this algorithm with the GCC compiler and the Clang compiler. From analysis of the elapsed time it took to run the TinyJAMBU algorithm on two different compilers, the Clang compiler proved to be slightly faster and more efficient than the GCC compiler.**

Keywords—cryptography, TinyJAMBU, C++, Python, compilers, performance

I. Introduction

TinyJAMBU is a lightweight cryptographic algorithm that is a variant of the JAMBU mode. The state and block sizes TinyJAMBU is significantly smaller than JAMBU. This algorithm supports key sizes of 128 bits, 192 bits, and 256 bits. The C++ implementation of the TinyJAMBU algorithm used in this paper was created by software engineer Anjan Roy [1], and is based off of the TinyJAMBU v2 implementation designed by Hongjun Wu and Tao Huang at the Division of Mathematical Sciences at Nanyang Technological University [2]. GCC is a collection of compilers in the GNU Compiler Collection, which is a set of compilers for multiple different programming languages, such as C, C++, Fortran, Ada, Go, and more. Clang is another compiler that supports languages in the C language family, such as C and C++[3]. The two compilers are two of the most prominent compilers for C languages today. This paper aims to see which compiler between them performs better with the TinyJAMBU cryptographic algorithm.

II. Methodology and Benchmarking Environment

A. Architecture and Setup

All benchmarking was done in a virtual environment. Utilizing Oracle's Virtual Box, this benchmark was performed in Ubuntu 20.04 LTS. The version of Clang used was version 10.0.0. The version of GCC-10 used was 10.5.0. The main computer hardware specifications include the following:

- 9th Gen Intel Core i7-9750H
- NVIDIA GeForce RTX 2070 Max-Q
- 16GB DDR4 System Memory
- 8GB GDDR6 Video Memory

Each test ran utilizing 6 cores of the Intel i7-9750H processor, at 2592.01 MHz per CPU. In the L1 cache memory, there were 32 KiB of data and 32KiB of instructions. The benchmark measured encryption and decryption execution times, CPU times, and iterations, based on variable lengths of cipher texts. The message block and the amount of feedback bits were a constant 32 bits for all tests.

Utilizing the Google Benchmark library [4], we're able to measure the elapsed times, the CPU times, and iterations. After setting up VirtualBox, utilizing the Ubuntu 20.04 LTS ISO, downloading the libraries for the C++ TinyJAMBU implementation as well as the Google Benchmark library, we're able to run the benchmark for the algorithm for keys of length 128-bit, 192-bit, and 256-bit, as shown in the following tables.

B. TinyJAMBU GCC Compiler Benchmark Results

TABLE I. 128-BIT KEY TINYJAMBU BENCHMARK, GCC COMPILER

TinyJAMBU GCC Compiler Benchmark, 128-bit Key				
Benchmark	***Cipher Text Length***	***Elapsed Time (ns)***	***CPU Time (ns)***	***Iterations***
Encrypt	64	71.2	71.1	9495955
Decrypt	64	75.5	75.4	7617103
Encrypt	128	84.3	84.3	7960551
Decrypt	128	89.2	89.1	7593124
Encrypt	256	116	116	6127276
Decrypt	256	119	119	5726532
Encrypt	512	180	180	2963753
Decrypt	512	175	175	3875714
Encrypt	1024	305	305	2449952
Decrypt	1024	289	289	2411235
Encrypt	2048	519	514	1357901
Decrypt	2048	515	515	1294626
Encrypt	4096	998	997	717637
Decrypt	4096	958	957	724850

Fig. 1. Table summary of the results of benchmarking the TinyJAMBU C++ implementation compiled in GCC, with a 128-bit key.

TABLE II. 192-BIT KEY TINYJAMBU BENCHMARK, GCC COMPILER

TinyJAMBU GCC Compiler Benchmark, 192-bit Key				
Benchmark	***Cipher Text Length***	***Elapsed Time (ns)***	***CPU Time (ns)***	***Iterations***
Encrypt	64	79.8	79.8	8199337
Decrypt	64	86.4	86.3	7316667
Encrypt	128	95.6	95.6	7123217
Decrypt	128	102	102	6725788
Encrypt	256	126	126	5428141
Decrypt	256	132	132	5569508
Encrypt	512	186	186	3731149
Decrypt	512	193	193	3646294
Encrypt	1024	300	300	2347730
Decrypt	1024	309	309	2249846
Encrypt	2048	518	517	1339384
Decrypt	2048	563	563	1216099
Encrypt	4096	985	984	693090
Decrypt	4096	994	993	702548

Fig. 2. Table summary of the results of benchmarking the TinyJAMBU C++ implementation compiled in GCC, with a 192-bit key.

TABLE III. 256-BIT KEY TINYJAMBU BENCHMARK, GCC COMPILER

TinyJAMBU GCC Compiler Benchmark, 256-bit Key				
Benchmark	***Cipher Text Length***	***Elapsed Time (ns)***	***CPU Time (ns)***	***Iterations***
Encrypt	64	73.0	73.0	8751660
Decrypt	64	76.9	76.8	9039270
Encrypt	128	88.2	88.2	7819704
Decrypt	128	91.4	91.4	7348778
Encrypt	256	118	118	6085791
Decrypt	256	120	120	4398762
Encrypt	512	186	186	3716075
Decrypt	512	176	176	3457891
Encrypt	1024	285	285	2329934
Decrypt	1024	290	290	2354627
Encrypt	2048	519	519	1259347
Decrypt	2048	529	529	1168170
Encrypt	4096	957	957	634665
Decrypt	4096	1067	1067	582095

Fig. 3. Table summary of the results of benchmarking the TinyJAMBU C++ implementation compiled in GCC, with a 256-bit key.

C. TinyJAMBU Clang Compiler Benchmark Results

TABLE IV. 128-BIT KEY TINYJAMBU BENCHMARK, CLANG COMPILER

TinyJAMBU Clang Compiler Benchmark, 128-bit Key				
Benchmark	***Cipher Text Length***	***Elapsed Time (ns)***	***CPU Time (ns)***	***Iterations***
Encrypt	64	70.8	70.8	9963749
Decrypt	64	79.8	79.7	8686099
Encrypt	128	102	91.8	8296536
Decrypt	128	90.9	90.9	8200675
Encrypt	256	110	110	6119417
Decrypt	256	123	122	5468743
Encrypt	512	175	175	3885993
Decrypt	512	215	214	3852183
Encrypt	1024	328	328	2384895
Decrypt	1024	330	330	2392181
Encrypt	2048	540	540	1365899
Decrypt	2048	546	545	1306265
Encrypt	4096	1015	1014	684962
Decrypt	4096	984	982	714019

Fig. 4. Table summary of the results of benchmarking the TinyJAMBU C++ implementation compiled in Clang, with a 128-bit key.

TABLE V. 192-BIT KEY TINYJAMBU BENCHMARK, CLANG COMPILER

TinyJAMBU Clang Compiler Benchmark, 192-bit Key				
Benchmark	***Cipher Text Length***	***Elapsed Time (ns)***	***CPU Time (ns)***	***Iterations***
Encrypt	64	93.7	93.6	8263304
Decrypt	64	92.8	92.7	7784718
Encrypt	128	96.3	96.3	6981783
Decrypt	128	109	109	6637518
Encrypt	256	134	134	4662549
Decrypt	256	139	139	4630137

TinyJAMBU Clang Compiler Benchmark, 192-bit Key				
Benchmark	***Cipher Text Length***	***Elapsed Time (ns)***	***CPU Time (ns)***	***Iterations***
Encrypt	512	191	191	3747377
Decrypt	512	193	193	3606906
Encrypt	1024	297	297	2281814
Decrypt	1024	307	306	2325037
Encrypt	2048	568	565	1324061
Decrypt	2048	537	537	1363393
Encrypt	4096	1042	1041	654262
Decrypt	4096	1093	1092	603619

Fig. 5. Table summary of the results of benchmarking the TinyJAMBU C++ implementation compiled in Clang, with a 192-bit key.

TABLE VI. 256-BIT KEY TINYJAMBU BENCHMARK, CLANG COMPILER

TinyJAMBU Clang Compiler Benchmark, 256-bit Key				
Benchmark	***Cipher Text Length***	***Elapsed Time (ns)***	***CPU Time (ns)***	***Iterations***
Encrypt	64	72.9	72.9	9474823
Decrypt	64	81.5	81.4	9142459
Encrypt	128	88.0	88.0	7305085
Decrypt	128	93.5	93.5	7391933
Encrypt	256	114	114	6052136
Decrypt	256	120	120	5709360
Encrypt	512	191	191	3931573
Decrypt	512	179	179	3357417
Encrypt	1024	289	289	2412178
Decrypt	1024	289	289	2386002
Encrypt	2048	563	562	1247786
Decrypt	2048	543	543	1143018
Encrypt	4096	1017	1016	594645
Decrypt	4096	1018	1018	606637

Fig. 6. Table summary of the results of benchmarking the TinyJAMBU C++ implementation compiled in Clang, with a 256-bit key.

III. ANALYSIS OF COMPILER BENCHMARK RESULTS

Utilizing the information gathered from the benchmark above, we can analyze the performance of each compiler and compare them. The benchmark gave us elapsed time, the CPU time, and the number of iterations for each benchmark. This allows us to use a few formulas for performance evaluation to help determine performance metrics and compare both compilers.

A. Relative Performance

The comparison between the performance of two different processors can be defined for this situation as:

$$Performance_A / Performance_B = n \quad (1)$$

Where A and B are the elapsed time of compilers, and n is a unitless factor of how much faster compiler A is than compiler B. Utilizing this formula on the above data, for when the GCC compiler elapsed time is compiler A and the Clang compiler elapsed time is compiler B, as well as vice versa, the results yield the following:

TABLE VII. GCC VS CLANG ELAPSED TIME COMPARISON

TinyJAMBU Factors of Speedup Utilizing GCC vs Clang				
Benchmark	***Cipher Text Length***	***128-bit Key***	***192-bit Key***	***256-bit Key***
Encrypt	64	1.006	0.852	1.001
Decrypt	64	0.946	0.931	0.944
Encrypt	128	0.826	0.993	1.002
Decrypt	128	0.981	0.936	0.978
Encrypt	256	1.055	0.940	1.035
Decrypt	256	0.967	0.950	1.000
Encrypt	512	1.029	0.974	0.974
Decrypt	512	0.814	1.000	0.983
Encrypt	1024	0.930	1.010	0.986
Decrypt	1024	0.876	1.007	1.003
Encrypt	2048	0.961	0.912	0.922
Decrypt	2048	0.943	1.048	0.974
Encrypt	4096	0.983	0.945	0.941
Decrypt	4096	0.974	0.909	1.048

Fig. 7. Table summary of the results of utilizing the relative performance equation with A being the elapsed time of the GCC compiler, and B being the elapsed time of the Clang compiler.

The average performance for keys of length 128-bit, 192-bit, and 256-bit are 0.949, 0.958, and 0.985 respectively. Applying the definition of the relative performance metric, this is an early indicator that the GCC compiler is slightly slower than the Clang compiler when utilizing this implementation of the TinyJAMBU algorithm, because this formula is applied on the elapsed times gathered on the benchmark for this algorithm.

To further support this point, the calculations are also performed with the Clang compiler elapsed time as A, and the GCC compiler elapsed time as B in Fig. 8 below.

TABLE VIII. CLANG VS GCC ELAPSED TIME COMPARISON

TinyJAMBU, Factors of Speedup Utilizing Clang vs GCC				
Benchmark	***Cipher Text Length***	***128-bit Key***	***192-bit Key***	***256-bit Key***
Encrypt	64	0.994	1.174	0.999
Decrypt	64	1.057	1.074	1.060
Encrypt	128	1.210	1.007	0.998
Decrypt	128	1.019	1.069	1.023
Encrypt	256	0.948	1.063	0.966
Decrypt	256	1.034	1.053	1.000
Encrypt	512	0.972	1.027	1.027
Decrypt	512	1.229	1.000	1.017
Encrypt	1024	1.075	0.990	1.014
Decrypt	1024	1.142	0.994	0.997
Encrypt	2048	1.040	1.097	1.085
Decrypt	2048	1.060	0.954	1.026
Encrypt	4096	1.017	1.058	1.063
Decrypt	4096	1.027	1.100	0.954

Fig. 8. Table summary of the results of utilizing the relative performance equation with A being the elapsed time of the Clang compiler, and B being the elapsed time of the GCC compiler.

The average performance for keys of length 128-bit, 192-bit, and 256-bit are 1.059, 1.047, and 1.016 respectively. This provides more evidence that the Clang compiler is faster utilizing the TinyJAMBU C++ implementation than the GCC compiler.

B. Challenges and Pitfalls

Originally, this experiment was also going to be performed between the Python language and the C++ language. However, the python implementation of the benchmark proved to be much less fruitful. After modification to the benchmark implementation of the Python algorithm, the benchmark yielded the results below. This benchmark was performed in the same operating system on the same machine as mentioned in Section II, Part A. It was ran in Python 3.10.0 in a Python virtual environment to utilize some of the functions in the Python implementation of the TinyJAMBU algorithm, as the global Python installation in this version of Ubuntu is Python 3.8.10.

TABLE IX. TINYJAMBU PYTHON IMPLEMENTATION BENCHMARK

TinyJAMBU Python Implementation				
Benchmark	***Key Length (bits)***	***Elapsed Time (ns)***	***CPU Time (ns)***	***Iterations***
Encrypt	128	23308	23283	26902
Decrypt	128	24036	24036	25828
Encrypt	192	24490	24490	25712
Decrypt	192	24690	24690	26563
Encrypt	256	23129	23125	26996
Decrypt	256	24227	24226	28043

Fig. 9. Table summary of the results of the Python implementation of the TinyJAMBU algorithm.

The data provided by this implementation already made it clear that the C++ implementation of the TinyJAMBU algorithm was much more optimized and ideal for usage. The time it took to encrypt/decrypt each key length is thousands of times more in the Python implementation than in the C++ implementation, regardless of C++ compiler chosen. This may be due to the fact the implementation of the Python version of the algorithm is done utilizing a wrapper, and a separate Application Programming Interface (API). It uses the C++ implementation of the algorithm for the logic, which may be an explanation for the noticeably higher elapsed times and CPU times. There is more time spent translating between the languages through the API. Thus, a comparison and analysis between separate compilers appeared to be a better option. The implementation for this algorithm was most likely meant to be primarily used in C++, so a more meaningful analysis would be to compare performance between two of the most widely used C++ compilers.

It is also hard to perform more analysis on the compilers due to the information available from the benchmark. It would be ideal to also utilize more metrics such as cycles per instruction (CPI), but analysis utilizing this metric is dependent on the number of instructions. The only mention of the specific number of instructions in the benchmark results is that there is 32KiB of instructions in the L1 cache memory. As this is given in an amount of kilobytes, rather than a specific number of instructions, utilizing analysis with the CPI is much harder without making assumptions.

Thus, the main analysis performed is the relative performance between the two compilers. The CPI is utilized to also retrieve the CPU time, which is given to us in the benchmark results regardless. Since we are also given the elapsed time, the relative performance equation is the most concrete metric performed without making assumptions in the calculation.

IV. CONCLUSION

The relative performance between both compilers shows that Clang is the preferred C++ compiler when utilizing the C++ implementation lightweight cryptographic TinyJAMBU algorithm. The benchmarking results, conducted on a virtual environment with a set of defined hardware specifications, showed that Clang exhibited slightly better performance

compared to GCC across various key sizes (128-bit, 192-bit, and 256-bit).

REFERENCES

[1] https://github.com/itzmeanjan/tinyjambu

[2] H. Wu, T. Huang, TinyJAMBU: A Family of Lightweight Authenticated Encryption Algorithms (Version 2), 2021, pp.3-38.

[3] M. Jun, “GCC vs. Clang/LLVM: An In-Depth Comparison of C/C++ Compilers,” unpublished.

[4] https://github.com/google/benchmark