IoT Security: Performance Evaluation of Grain, MICKEY, and Trivium - Lightweight Stream Ciphers

Levent Ertaul, Arnold Woodall
CSU East Bay, Hayward, CA, USA
levent.ertaul@csueastbay.edu, awoodall@horizon.csueastbay.edu

Abstract - *In this paper, we evaluate the software implementation of eSTREAM Profile II finalists (Grain, MICKEY, and Trivium) on a NodeMCU development kit 1.0 microcontroller. The NodeMCU is programmed by Arduino IDE to run a C++ code that awaits TCP communication over a WiFi network to encrypt or decrypt text using these lightweight stream ciphers. Throughput performance of the cipher implementations on the NodeMCU device is measured and compared, as well as the overall WiFi network roundtrip execution time. Using these performance figures, we consider the suitability of these cipher algorithms as software implementations in IoT applications. Additionally, we consider the software and hardware implementation tradeoffs, as well as the device's power consumption running the stream ciphers.*

Keywords: IoT Security, Grain, MICKEY, Trivium, Microcontroller

1 Introduction

The proliferation of Internet of Things (IoT) devices in homes and businesses today have led to many concerns about the security of these typically small, resource constrained devices. [1] [2] Examples of resource constrained home devices may include kitchen appliances or sprinkler systems. Examples in businesses may include Radio Frequency Identification (RFID) security card readers, or industrial plant control systems. These devices have evolved in the past decade from disconnected, independent operating entities to becoming increasingly connected with other nearby electronic devices in a local area network, or even being accessible remotely across the Internet. Unlike a personal computer or smartphone, which typically runs a widely used and robust operating system on powerful hardware, IoT devices may be run by a microcontroller with low operating resources. But it is not always feasible for such a device to accomplish its intended function while simultaneously having sufficient resources to encrypt or decrypt data securely. We will introduce some published lightweight cryptographic algorithms, designed for IoT scale devices, to achieve the goal of secure data transfer in a resource constrained environment.

In August 2016, the National Institute of Standards and Technology (NIST) issued a report on the current state of lightweight cryptography. Lightweight cryptography is a subfield of cryptography that is concerned with algorithm performance in resource constrained devices. [3] The paper discusses the latest known secure lightweight cryptographic primitives available, divided into four different categories. One of these categories specifically discusses the current state of lightweight stream cipher primitives. The specific algorithms the paper references are finalists from the 2004-2008 ECRYPT Stream Cipher Project (eSTREAM) competition run by the European Commission [4]. The finalists were divided into two categories, Profile I and Profile II. Profile I stream cipher algorithms are designed for software applications with high throughput requirements. Profile II algorithms are designed for hardware applications where minimizing the number of gates or power consumption is important. The three specified stream ciphers in the NIST paper (Grain, MICKEY, and Trivium) are classified as Profile II.

Almost 10 years has passed since the eSTREAM finalists were first revealed and the technology landscape has continued to evolve, particularly in embedded systems. Previous papers have evaluated eSTREAM ciphers on FPGA devices or 8-bit AVR microcontrollers running at 8 MHz [5] [6]. In this paper, we will evaluate some of these eSTREAM ciphers on a modern microcontroller with compact dimension and low cost but increased processing and storage capability than previous devices considered. Section 2 will introduce lightweight stream ciphers and the eSTREAM Profile II portfolio. Section 3 will describe the software implementation and testing framework. Section 4 will review the results obtained from testing and some important observations. Section 5 will offer our final thoughts on these stream ciphers for IoT applications.

2 Lightweight Stream Ciphers

The operation of a stream cipher, specifically a synchronous stream cipher, can be generally summarized by the high-level block diagram in Figure 1 [7]. There is a keystream generator block that takes as input a symmetric key and initialization vector (IV). The key is intended to be a privately known quantity, while the IV is public. When the system is run, it computes one keystream bit per cycle. If text

is being encrypted then each bit of keystream is XORed with one bit of plaintext, to produce the resulting ciphertext. The encryption and decryption steps must use the same key and IV to encode and decode text successfully.

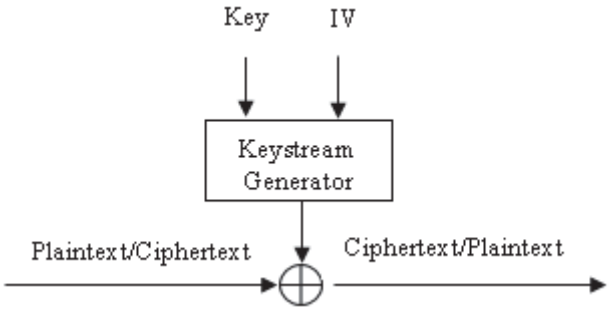

Figure 1 - Stream Cipher

A stream cipher is a generalization of a one-time pad, where its security depends on the pseudo randomness of the keystream generator [8]. One of the primary characteristics advertised by a secure stream cipher implementation is the nonrepeatable keystream length. Another important characteristic of a secure stream cipher is the bit length of the symmetric key used. The keystream generator will typically be comprised of a combination of Linear Feedback Shift Registers (LFSRs) and Non-Linear Feedback Shift Registers (NFSRs) [8]. The registers will have feedback provided by functions that combine data from select register taps using XOR or AND operations. A global clock running at a specific frequency will determine the speed that the cipher executes at. A lightweight stream cipher possesses these stream cipher characteristics, and is also designed to minimize space and computational time requirements. We will now review the specifications of the lightweight stream cipher finalists from eSTREAM (Profile II): Grain, MICKEY, and Trivium.

2.1 Grain

The Grain stream cipher [9] uses a fixed 80-bit key and 64-bit IV. Its keystream generator is composed of an LFSR and NFSR as well as three primitive polynomial functions that determine the feedback and output bits. The system initially loads the IV and key into the LFSR and NFSR, respectively, and then starts to generate usable keystream after 160 rounds of cycling the system which includes feeding back the output back into the registers. It can generate a maximum of 2^80 bits of unique keystream. Figure 2 is a diagram of Grain's architecture [9].

Additionally, an enhanced Grain stream cipher supporting a 128-bit key is also published [10]. The 128-bit Grain design is very similar to 80-bit Grain. Some notable differences include the introduction of a 128-bit LFSR, 128-bit NFSR, a 96-bit IV, 256 initialization cycles, and modified feedback functions using updated primitive polynomials.

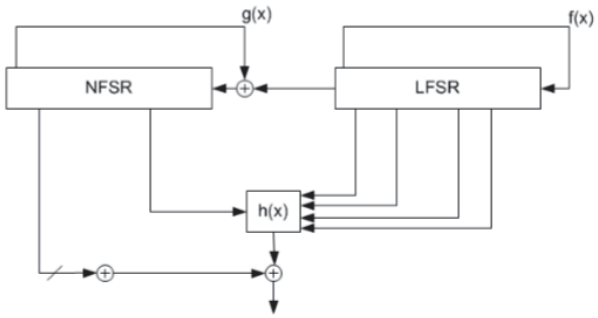

Figure 2 – Grain (80-bit)

2.2 MICKEY

The Mutual Irregular Clocking Keystream (MICKEY) stream cipher [11] uses a fixed 80-bit key and a 0 to 80-bit IV. MICKEY's keystream generator is composed of two 100-bit shift registers, where each register is irregularly clocked by the outputs of both registers. After the key and IV are loaded, the system is then cycled 100 times before generating usable keystream. Each key and IV pair can produce up to 2^40 bits of unique keystream. Figure 3 is a diagram of MICKEY's variable clocking architecture [11].

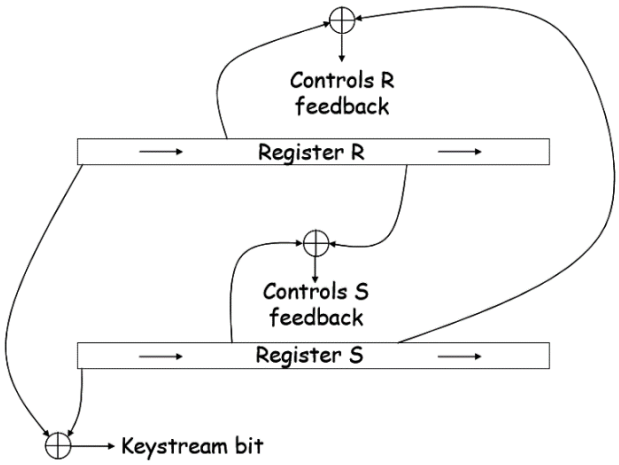

Figure 3 – MICKEY

An upgraded version of MICKEY that supports 128-bit keys is also available [12]. The 128-bit MICKEY architecture is quite similar to the 80-bit edition. The 128-bit edition supports an IV up to 128 bits in length and updated feedback tap locations.

2.3 Trivium

The Trivium stream cipher [13] uses an 80-bit key and 80-bit IV. The keystream generator contains a 288-bit long register, where specific registers are read and fed back into the system as it is clocked in a circular pattern (i.e. the system rotates as it is cycled.) The key is loaded in the first 93 bits and

the IV is loaded in the next 84 bits. The entire system is then cycled 4 * 288 times before usable keystream is available. The system can produce up to 2^64 bits of unique keystream. Figure 4 is a diagram of Trivium [5].

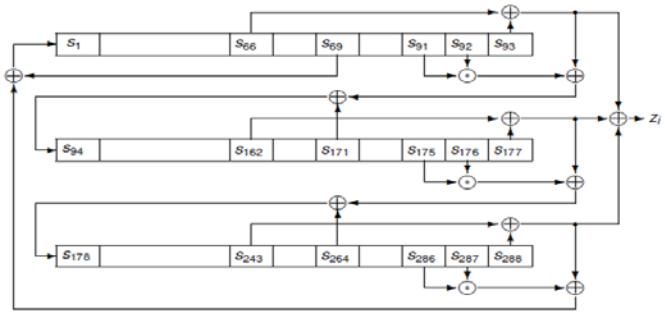

Figure 4 – Trivium

Unlike Grain and MICKEY, the authors of Trivium did not submit a 128-bit version of the cipher to eSTREAM.

2.4 Known Security Attacks

Each of these lightweight stream cipher algorithms have undergone rigorous cryptanalysis during the eSTREAM competition to emerge as a finalist candidate [14]. However published research in years since the end of the eSTREAM competition has shown new potential vulnerabilities in each stream cipher to be aware of. Security attacks are varied in approach and take advantage of flaws in the cipher algorithm design to help reduce the time to decrypt a ciphertext. For example, a Differential Fault Analysis Attack on a hardware implementation of a stream cipher is possible by injecting faults in the cipher state and analyzing a faulty and fault-free ciphertext to deduce the internal cipher state or key [15]. Table 1 lists some known security attacks on the eSTREAM finalists that have been published by the research community.

Cipher	Known Security Attacks
Grain	Differential Fault Analysis Attack [15] Conditional Differential Attack [16]
MICKEY	Meet-In-The-Middle Attack [17] Differential Fault Analysis Attack [18]
Trivium	Cube Attack [19] Scan-based Attack [20]

Table 1 – Stream Cipher Attack Vectors

3 NodeMCU devkit Implementation of Grain, MICKEY, and Trivium

We decided to test these ciphers on a NodeMCU development kit (devkit) 1.0 microcontroller, which is a recently published open source IoT platform [21]. Onboard the microcontroller is an ESP8266-based WiFi module produced by AI Thinker (vendor code ESP-12E). The device specifications are listed in Table 2 [22] [23].

Device Feature	Specification
CPU	80 MHz 32-bit
Dynamic RAM	128 KB
Flash Memory	4 MB
WiFi	2.4 GHz 802.11 b/g/n
Serial I/O	Micro USB

Table 2- NodeMCU devkit 1.0 Specifications

A pair of NodeMCU devkit modules for this experiment were obtained from Amazon.com, which cost around 15 dollars for the pair. The development kit features many I/O pinouts for component interfacing, however for this experiment we did not use these capabilities. The microcontroller program developed for this experiment only uses the WiFi element for TCP data transfer, and the Micro USB connection for printing console statements with program state information and reporting results.

The NodeMCU devkit device was connected to a Dell Inspiron 7368 laptop running Microsoft Windows 10 Professional via Micro USB cable. To program the device and monitor its activity, the Arduino IDE v1.8.1 was installed on the laptop which allows a C++ code to be compiled for the microcontroller. A special NodeMCU configuration had to be installed and loaded in the Arduino IDE to be able to target the specific device used [24]. The selected configuration, "NodeMCU 1.0 (ESP-12E Module)", allows for 80 MHz CPU, 1MB of flash memory for the program code and 3MB for filesystem, 80 kB data RAM, and 32 kB instruction RAM [25].

The program uploaded to the NodeMCU devkit's flash memory runs WiFi initialization code for the ESP-12E chip to target the laptop's mobile WiFi hotspot and establish network connectivity. It then opens a dedicated TCP socket on port 80 that listens for incoming encryption or decryption requests. Depending on the bytes passed in the request, the requested algorithm would be executed with associated configuration information. The requests had to follow the message header pattern in Table 3, followed by the plaintext or ciphertext to process. An 80-bit key for each 80-bit cipher was hardcoded into the NodeMCU application code. Likewise, a 128-bit key for each 128-bit cipher was hardcoded.

Byte	Purpose
1	Algorithm (Grain, MICKEY, Trivium)
2	Mode (Encrypt, Decrypt, Test)
3	Network Option (Variable or Fixed Response Packet Sizes)
4	IV Length in Bits
5-6	Block Size in Bytes
7-10	Text Length in Bytes
11-?	IV bytes (Byte length depends on byte 4 value)

Table 3 - Stream Cipher Server TCP Message Header

The lightweight stream cipher algorithm code used in the microcontroller program was based on reference implementations originally submitted to eSTREAM [26] and modified to run on NodeMCU. Initially the unoptimized reference code of Grain and MICKEY were tested, alongside the Trivium implementation (only one Trivium implementation was provided by the author). A performance optimized version of Grain and MICKEY was subsequently used for final evaluation. When testing the optimized Grain cipher provided to eSTREAM, it did not successfully run on the NodeMCU because of memory alignment issues when dereferencing pointer addresses. As substitute we attempted optimization of the eSTREAM Grain reference cipher code to achieve improved throughput over the reference version. To validate the ciphers were working correctly, a test mode was added to the program to verify a certain key and IV produced a test vector that is identical to the reference test vector provided by the author of the respective cipher.

```
for (int i = 0; i < data.Length; i += WRITESIZE)
{
    bytesnotsent = data.Length - i;
    if (transitbytes < REMOTEBUFFERSIZE)
    {
        if (bytesnotsent < WRITESIZE)
        {
            stream.Write(data, i, bytesnotsent);
            transitbytes += bytesnotsent;
        }
        else
        {
            stream.Write(data, i, WRITESIZE);
            transitbytes += WRITESIZE;
        }
    }
    else
    {
        i -= WRITESIZE;
    }

    if (stream.DataAvailable)
    {
        int inbytes = stream.Read(retbyte, 0, READSIZE);
        if (inbytes != 0)
        {
            transitbytes -= inbytes;
            retbyteleft -= inbytes;
        }
    }
}

while (retbyteleft > 0)
{
    int inbytes = stream.Read(retbyte, 0, READSIZE);
    if (inbytes != 0)
    {
        transitbytes -= inbytes;
        retbyteleft -= inbytes;
    }
}
```

Figure 5 – .Net TCP data transfer

On the Dell laptop, a Microsoft .Net client application was developed to interface with NodeMCU. The application is a TCP socket client and allows for either an interactive mode and batch job mode. The application fills out the message pattern in Table 2 to execute the encryption or decryption. The plaintext or ciphertext to be processed is read in from a locally specified text file. To avoid overfilling the microcontroller's TCP buffers, only 2 full packets of data were allowed in-flight to the NodeMCU at a time. When a packet returned with data from the NodeMCU then another would be allowed to be sent if the total number of bytes outstanding doesn't exceed the maximum allowed outstanding. Figure 5 shows the .Net client code used to manage the network data processing (timing code and console print statements removed for brevity). The TCP result responses containing the plaintext or ciphertext would be written to an output file. The application continuously tracks how many bytes are left outstanding. Once all bytes have been processed and returned the final round trip time is recorded to a log file.

4 Performance Results

Automated tests were run using the .Net client program running on the Dell laptop. Tests were performed for each cipher, for multiple text lengths, and different block sizes (the quantity of data processed by a single call to encrypt). An executed test would output data to the NodeMCU serial I/O console, similar to the output shown in Figure 6.

```
Grain
Encrypt Mode
Variable Write Packet Sizes (no memcpy)
IV length (bits): 64
Read length (bytes): 1460
Text length (bytes): 1000000
IV: 00000000
getFreeHeap: 36136
algMicros: 30048123
algCycles: 2404099110
```

Figure 6 - Example of NodeMCU devkit Serial Output

In the serial output, the value of primary interest is algMicros which represents the total time in microseconds executing the targeted cipher on just the microcontroller. A compilation of algMicros results was recorded and compared in Excel. algCycles, which represents the number of clock cycles elapsed, was also recorded. The value of algCycles returned in each test approximated the same value as algMicros multiplied by 80, since the CPU clock runs at a frequency of 80 times per microsecond, or 80 MHz. In addition, a timing measurement was taken by the client application for the full round trip time to process all the text and receive the full response back. A compilation of these roundtrip times was also logged into Excel. The timings were then converted into throughput values which are finally compared across ciphers.

4.1 Memory Utilization

The required space for our compiled Arduino sketch with WiFi library, TCP server, and all cipher algorithms was 236KB out of 1MB user programmable space (a blank sketch requires 216KB). Each stream cipher implemented required an additional 2-3KB of additional flash space. In regards to RAM usage, global variables required 40KB out of the 80KB of RAM available (a blank sketch required 30KB). Each additional stream cipher implemented required an additional 1-2KB of RAM for global variables. While ample flash space remains available for an IoT application, the limited quantity of available RAM will limit the types of IoT applications that can be developed to run on the NodeMCU devkit. For example, in a webcam application the device will likely not be able to process an entire high resolution photo in memory. Streaming data through the device for incremental processing helps manage RAM utilization.

4.2 Lightweight Stream Cipher Throughput

The primary observation made after running all stream cipher algorithms is that Trivium outperformed the other cipher algorithm implementations in throughput by at least two orders of magnitude. Figure 7 shows the dominance of Trivium algorithm performance as compared to the throughput of other eSTREAM ciphers when processing a 1MB plaintext file.

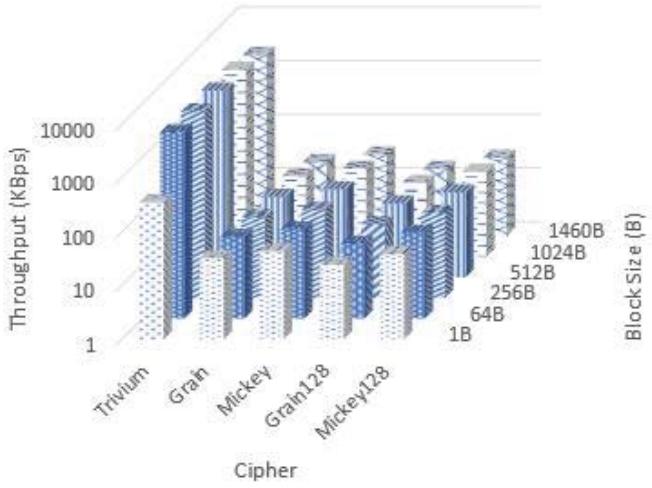

Figure 7 – Stream Cipher Throughput on NodeMCU

A second observation of Figure 7 results is that changing block size had negligible impact, which can also be seen in Figure 8. An increased block size tends to only reduce the number of overall invocations of the cipher algorithm to produce keystream bits, but does not change the number of internal algorithm steps required in total to generate each keystream bit.

One final observation regarding Figure 7 is that the amount of total algorithm time increased in direct proportion to the total length of text processed. For example, increasing the amount of text processed by a factor of 10 leads to an increase in processing time by approximately a factor of 10 as well. This implies that processing time for each keystream bit is fairly constant.

4.3 WiFi Round Trip Throughput

Figure 8 shows a comparison of each algorithm's overall WiFi throughput in the case of reporting back 1MB of data using variable sized packets (i.e. not every packet returned had a full TCP payload of Maximum Segment Size, or MSS, of 1460B). Compared to Figure 7, the throughput of executing Trivium when factoring in WiFi communication is significantly decreased by two orders of magnitude. Trivium's WiFi roundtrip throughput was only about 2-3 times better than all other stream ciphers, as Grain and MICKEY incurred only a slight decrease in WiFi roundtrip throughput performance.

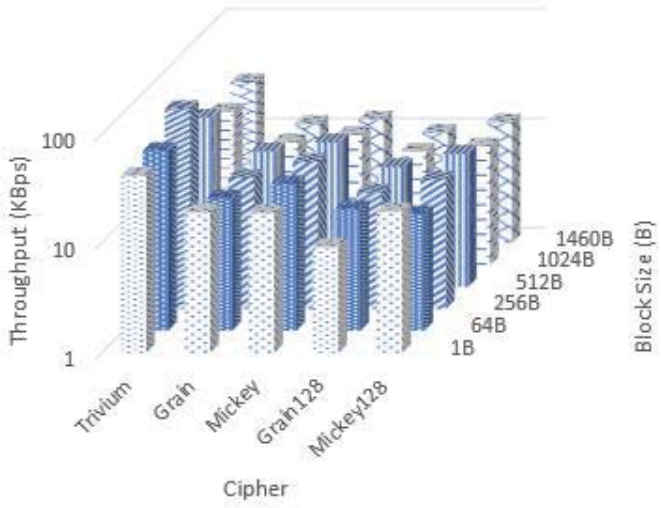

Figure 8 – WiFi Throughput using Variable Size Packets

Figure 9 shows a similar comparison of WiFi throughput when processing a 1MB file as Figure 8 does. However, in this case we attempt to only send back results in full packets of MSS length. However, to send a full packet a linear buffer had to equal or exceed the size of MSS. A memcpy operation was required to move the leftover bytes to the beginning of the buffer if the buffer content exceeded MSS. Comparing Figure 8 and Figure 9 results, the performance of fixed length packets with memcpy was not significantly different than that of variable length packets.

A significant factor affecting the WiFi throughput during the experiment were random delays in long running data transfers, typically between 2 to 11 seconds per delay. A cause for the delay could not be definitively isolated, but it is expected some delay will occur when operating in a wireless

environment, especially when operating under a continuous workload. Maintenance operations on the microcontroller to reorganize TCP buffers and service interrupts may be a plausible explanation for the delay.

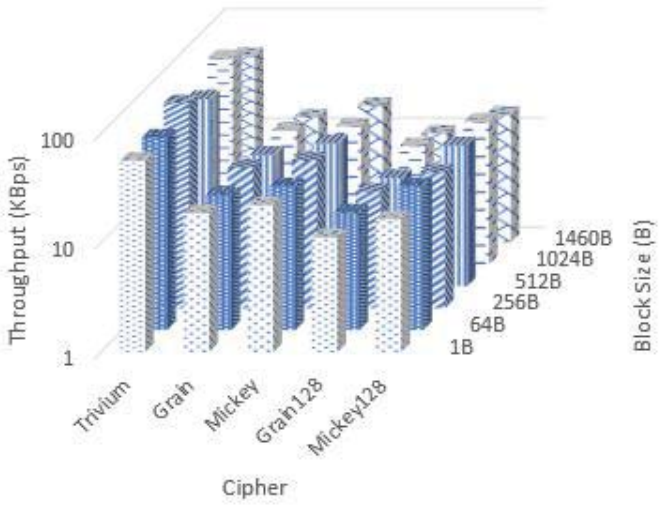

Figure 9 – WiFi Throughput using Fixed Size Packets

4.4 Hardware Implementation Comparison

Grain, MICKEY, and Trivium were all designed for hardware implementation because of the minimal amount of physical space each would take to implement their design [27]. However, it is still possible to run these algorithms in software with decent or excellent performance, depending on the requirements of the application. In a software implementation, additional clock cycles will be required, since one line of C++ code may translate into many lines of compiled assembly code. Software implementation performance is therefore expected to be poor when compared to a theoretical FPGA or ASIC hardware implementation of similar clock speed.

Consider a bit-by-bit FPGA hardware implementation of a stream cipher algorithm where each clock cycle produces one bit of keystream. An 80 MHz clocked hardware implementation could produce a throughput of 80 Mbps, or 10,000 KBps, of keystream. The fastest stream cipher implementation in our tests, Trivium, executed with a maximum throughput around 3,333 KBps. The Grain and MICKEY implementations performed at less than 50 KBps.

While the performance of microcontroller stream cipher implementation is less performant than FPGA or ASIC equivalents, it has many advantages. Microcontroller code can be revised and updated quickly to fix a flaw, add a feature, or improve performance. One can avoid having to fabricate a new part, or revise component spacing and timing requirements. The higher-level C++ code can even be made portable for deployment to other platforms. The cost of a microcontroller is generally cheaper than FPGA. Lastly it should be noted that side-channel security attacks that monitor power levels on FPGA implementations are not as easily possible with a microcontroller implementation due to the power noise emitted by its multiple onboard components [15].

4.5 Power Consumption

The power consumption of the stream ciphers running on NodeMCU devkit can be estimated by Equation 1 [28].

$$E = V_{cc} * \mathrm{I} * N * \tau \quad (1)$$

For NodeMCU devkit's ESP8266, the specifications [22] yield the following formula values. The value of algCycles is provided by the NodeMCU serial monitor output when testing each stream cipher's performance.

$$V_{cc} = 3.3\ V \quad (2)$$

$$I = .08\ A \quad (3)$$

$$N * \tau = algCycles * 80e^{-6}\ s \quad (4)$$

Table 4 lists the calculated power consumption for all ciphers running on NodeMCU devkit using a block size of 1460B and 1MB data file.

Trivium	Grain	Mickey	Grain128	Mickey128
0.081 J	7.933 J	5.749 J	10.599 J	6.559 J

Table 4 – Calculated Power Consumption

5 Conclusion

In this paper, we have evaluated the performance of a C++ software implementation of lightweight stream ciphers on NodeMCU devkit. Our findings show that the Trivium implementation provided the best throughput performance and lowest power consumption of all the stream ciphers evaluated. However, when evaluating the WiFi roundtrip performance, the throughput of Trivium significantly decreased but remained the best overall. We can see that even though a cipher implementation can be optimized for better algorithm throughput, external factors such as network congestion or other interrupts can mask these improvements. For example, transferring data with the NodeMCU WiFi module may be interrupted by more powerful nearby radio emissions. Also, the dynamic memory constraints of the NodeMCU devkit limits the amount of data that can be buffered for processing to smooth out external delays. Taking into account known security attacks, all of the evaluated stream ciphers in this paper are sufficient for adding data security to ESP8266 WiFi IoT applications with low to moderate data throughput requirements such as periodic sensor readings.

6 References

[1] J. Wurm, K. Hoang, O. Arias, A.-R. Sadeghi and Y. Jin, "Security Analysis on Consumer and Industrial IoT Devices," in *Design Automation Conference (ASP-DAC), 2016 21st Asia and South Pacific*, 2016.

[2] R. Mahmoud, T. Yousuf, F. Aloul and I. Zualkernan, "Internet of things (IoT) security: Current status, challenges and prospective measures," in *Internet Technology and Secured Transactions (ICITST)*, 2015.

[3] A. K. McKay, L. Bassham, M. S. Turan and N. Mouha, "Report on Lightweight Cryptography," NIST, 2016.

[4] "The eSTREAM portfolio," 2004. [Online]. Available: http://www.ecrypt.eu.org/stream/. [Accessed Jan 2017].

[5] A. Jafarpour, A. Mahdlo, A. Akbari and K. Kianfar, "Grain and Trivium Ciphers Implementation Algorithm in FPGA Chip and AVR Micro Controller," in *ICCAIE*, 2011.

[6] G. Meiser, T. Eisenbarth, K. Lemke-Rust and C. Paar, "Efficient Implementation of eSTREAM Ciphers on 8-bit AVR Microcontrollers," *Industrial Embedded Systems,* 2008.

[7] C. Paar, "Lecture 3: Stream Ciphers, Random Numbers and the One Time Pad," 30 Jan 2014. [Online]. Available: https://www.youtube.com/watch?v=AELVJL0axRs. [Accessed Feb 2017].

[8] M. Stamp, "Stream Ciphers," [Online]. Available: http://www.cs.sjsu.edu/~stamp/crypto/PowerPoint_PDF/5_StreamCiphers.pdf. [Accessed Feb 2017].

[9] M. Hell, T. Johansson and W. Meier, "Grain - A Stream Cipher for Constrained Environments," 2005. [Online]. Available: http://www.ecrypt.eu.org/stream/p3ciphers/grain/Grain_p3.pdf. [Accessed Jan 2017].

[10] M. Hell, T. Johansson and A. Maximov, "A Stream Cipher Proposal: Grain-128," ISIT, Seattle, 2006.

[11] S. Babbage and M. Dodd, "The stream cipher MICKEY 2.0," 2006. [Online]. Available: http://www.ecrypt.eu.org/stream/p3ciphers/mickey/mickey_p3.pdf. [Accessed Jan 2017].

[12] S. Babbage and M. Dodd, "The stream cipher MICKEY-128 2.0," 2006. [Online]. Available: http://www.ecrypt.eu.org/stream/p3ciphers/mickey/mickey128_p3.pdf. [Accessed Jan 2017].

[13] C. De Cannière and B. Preneel, "Trivium specifications," 2005. [Online]. Available: http://www.ecrypt.eu.org/stream/p3ciphers/trivium/trivium_p3.pdf. [Accessed Jan 2017].

[14] S. Babbage, C. De Canniere, A. Canteaut, C. Cid, H. Gilbert, T. Johansson, M. Parker, B. Preneel, V. Rijmen and M. Robshaw, "The eSTREAM Portfolio," 15 April 2008. [Online]. Available: http://www.ecrypt.eu.org/stream/portfolio.pdf. [Accessed Feb 2017].

[15] A. Chakraborty, B. Mazumdar and D. Mukhopadhyay, "A Combined Power and Fault Analysis Attack on Protected Grain Family of Stream Ciphers," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. PP, no. 99, 2017.

16] Z. Ma, T. Tian and W.-F. Qi, "Improved conditional differential attacks on Grain v1," *IET Information Security,* vol. 11, no. 1, pp. 46-53, 2017.

[17] T. Helleseth, C. Jansen, O. Kazymyrov and A. Kholosha, "State Space Cryptanalysis of The MICKEY Cipher," in *Information Theory and Applications Workshop (ITA)*, 2013.

[18] S. Karmarkar and D. Chowdhury, "Differential Fault Analysis of MICKEY-128 2.0," in *Workship on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, 2013.

[19] S. Islam and I. Ul Haq, "Cube Attack on Trivium and A5/1 Stream Ciphers," in *13th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, 2016.

[20] M. Fujishiro, M. Yanagisawa and N. Togawa, "Scan-based attack against Trivium stream cipher independent of scan structure," in *10th International Conference on ASIC*, 2013.

[21] "NodeMCU," [Online]. Available: http://www.nodemcu.com/index_en.html. [Accessed Feb 2017].

[22] "ESP-12E WiFi Module," [Online]. Available: https://mintbox.in/media/esp-12e.pdf. [Accessed Jan 2017].

[23] "NodeMCU," [Online]. Available: https://en.wikipedia.org/wiki/NodeMCU. [Accessed Feb 2017].

[24] "Quick Start to Nodemcu (ESP8266) on Arduino IDE," [Online]. Available: http://www.instructables.com/id/Quick-Start-to-Nodemcu-ESP8266-on-Arduino-IDE/. [Accessed Jan 2017].

[25] "Memory Map esp8266," [Online]. Available: https://github.com/esp8266/esp8266-wiki/wiki/Memory-Map. [Accessed Feb 2017].

[26] "Repository," [Online]. Available: http://www.ecrypt.eu.org/stream/svn/viewcvs.cgi/ecrypt/trunk/submissions/. [Accessed Jan 2017].

[27] S. Babbage, J. Borghoff and V. Velichkov, "The eSTREAM Portfolio in 2012," 2012. [Online]. Available: http://www.ecrypt.eu.org/ecrypt2/documents/D.SYM.10-v1.pdf. [Accessed Feb 2017].

[28] D. Salama, H. A. Kader and M. Hadhoud, "Studying the Effects of Most Common Encryption Algorithms," *International Arab Journal of e-Technology,* vol. 2, no. 1, 2011.