XTEA Encryption comparison using Arduino Uno and Raspberry Pi 3 B+ (Spring 2021)

Isaac Elizarraz, Gene Drumheller, Aaron Nguyen, Mohamed El-Hadedy
ECE-Department, College of Engineering, C alifornia State Polytechnic University, Pomona
Email: { ielizarraz, Grdrumheller, aaronnguyen, **mealy }** @cpp.edu

***Abstract*—This article explains the basic description and usage of the XTEA Encryption, developed by David Wheeler and Roger Needham in 1997, and the results of gauging the power usage and efficiency of the algorithms. The experiment was conducted on Arduino Uno using the C language and Raspberry Pi 3 B+ using the Python 2 language. The experiment has found the code under Arduino to run with less than half the power usage than the Raspberry Pi.**

I. INTRODUCTION

THE XTEA algorithm is the successor to the TEA algorithm, which XTEA uses a 64 bit block Feistel cipher with a 128-bit key and suggested 64 rounds. The cipher was designed by David Wheeler and Roger Needham of the Cambridge Computer Laboratory and the algorithm was first presented in 1997. Wheeler and Needham are both responsible for the development of TEA, XTEA, and its successors the XXTEA. Their goal was to develop a simple and short cipher that does not rely on tables or pre-computations. Each successor was made to correct a weakness from the predecessor. For example, TEA suffers from equivalent keys, meaning each key is equivalent to three others, so even though the key is supposed to be 128-bits, TEA only effectively uses only 126 bits. XTEA is presented along with the variable-width block cipher term known as Block TEA, which uses XTEA round function. XXTEA was developed to correct the weakness in Block TEA. The differences between the TEA and XTEA algorithms are stark, where the XTEA contains a more complex key-schedule as well as a rearrangement of the shift, XORs, and additions in the traditional combinational logic instantiation. The encryption is not subject to any patents.

II. RELATED WORK

An article by Jens-Peter Kaps at the George Mason University, Fairfax, VA has done a similar work using the XTEA algorithm, and implementing the code onto a FPGA. The goal was to create an ultra-low power implementation, which he concluded that it consumes less energy than compared to the AES implementation. The codes were written in VHDL instead of C or Python.

III. IMPLEMENTATION

Because the key requires 128 bits, the key must be generated with 16 characters (16 times 8 bits). As the figure 1 and figure 2 below shows, in both the standard C source code and our Python implementation, we have four main variables (v0, v1, sum, and delta). The delta and masks are in 64 bits. Logical left shift of 4 are denoted as << 4 and logical right shift is denoted as >> 5. The bitwise XOR function is denoted as ^. The for loop should occur 32 times, which is hard-coded in the python code and is typically passed down in the C code. There are two methods or functions, one for the encryption and the other for decryption. The encryption and decryption are sort of like an inverse of each other, where the encryption is done by addition and decryption are done by subtraction. Figure 3 shows the simplified block diagram of XTEA algorithm.

```c
#include <stdint.h>

/* take 64 bits of data in v[0] and v[1] and 128 bits of key[0] - key[3] */

void encipher(unsigned int num_rounds, uint32_t v[2], uint32_t const key[4]) {
    unsigned int i;
    uint32_t v0=v[0], v1=v[1], sum=0, delta=0x9E3779B9;
    for (i=0; i < num_rounds; i++) {
        v0 += (((v1 << 4) ^ (v1 >> 5)) + v1) ^ (sum + key[sum & 3]);
        sum += delta;
        v1 += (((v0 << 4) ^ (v0 >> 5)) + v0) ^ (sum + key[(sum>>11) & 3]);
    }
    v[0]=v0; v[1]=v1;
}

void decipher(unsigned int num_rounds, uint32_t v[2], uint32_t const key[4]) {
    unsigned int i;
    uint32_t v0=v[0], v1=v[1], delta=0x9E3779B9, sum=delta*num_rounds;
    for (i=0; i < num_rounds; i++) {
        v1 -= (((v0 << 4) ^ (v0 >> 5)) + v0) ^ (sum + key[(sum>>11) & 3]);
        sum -= delta;
        v0 -= (((v1 << 4) ^ (v1 >> 5)) + v1) ^ (sum + key[sum & 3]);
    }
    v[0]=v0; v[1]=v1;
}
```

Fig. 1. Standard C Programming Implementation of XTEA Encryption

```python
def xtea_encrypt(key,block,n=32,endian="!"):

    v0,v1 = struct.unpack(endian+"2L",block)
    k = struct.unpack(endian+"4L",key)
    sum,delta,mask = 0L,0x9e3779b9L,0xffffffffL
    for round in range(n):
        v0 = (v0 + (((v1<<4 ^ v1>>5) + v1) ^ (sum + k[sum & 3]))) & mask
        sum = (sum + delta) & mask
        v1 = (v1 + (((v0<<4 ^ v0>>5) + v0) ^ (sum + k[sum>>11 & 3]))) & mask
    return struct.pack(endian+"2L",v0,v1)

def xtea_decrypt(key,block,n=32,endian="!"):

    v0,v1 = struct.unpack(endian+"2L",block)
    k = struct.unpack(endian+"4L",key)
    delta,mask = 0x9e3779b9L,0xffffffffL
    sum = (delta * n) & mask
    for round in range(n):
        v1 = (v1 - (((v0<<4 ^ v0>>5) + v0) ^ (sum + k[sum>>11 & 3]))) & mask
        sum = (sum - delta) & mask
        v0 = (v0 - (((v1<<4 ^ v1>>5) + v1) ^ (sum + k[sum & 3]))) & mask
    return struct.pack(endian+"2L",v0,v1)
```

Fig. 2. XTEA implementation using Python Programming

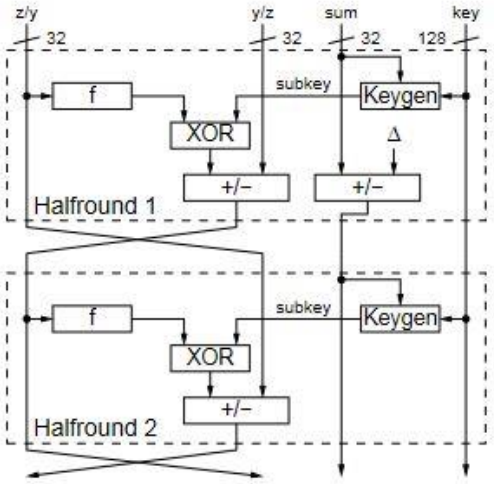

Fig. 3. Simple Block Diagram of XTEA Encryption

IV. EXPERIMENT

Our peers conducted research on the predecessor as well as the successors to our algorithm in order to gauge the power usage and efficiency of the algorithms of not only in C, but also python and other languages as well as across different microcontrollers.

In our experience working with the algorithm we first sourced it for C to be programmed onto an Arduino Uno. This originally came with the intention of testing encryption and decryption across the Arduino Uno and Raspberry Pi 3 B+ through transceivers. Unfortunately due to time constraints and the schedules of our team we did not have the time to do so, so we decided to test the algorithms in a controlled environment. We wrote the code and programmed the code onto the Arduino Uno to be run on a power meter and constantly running the same decryption and encryption process that the Pi would be running with no external accessories connected to either the microcontroller or the micro processing unit. This would allow our team to successfully deduce which unit would be using more power running the same rudimentary processes that are required to be run on each as well as the decryption/encryption code.

It came to our attention initially that because the Arduino is a simple microcontroller, and the Raspberry Pi is technically a microprocessor/microcontroller hybrid that powers an entire OS with multiple connectivity options, that one would definitely be less power efficient than the other. This was only a surmise prior to testing both of them with the processes, but was further confirmed after testing with the power meter. The Arduino uno managed to run the same algorithm with less than 50% of the power usage of the Pi. This is with all the other Pi's processes halted including WIFI and all I/O devices disconnected. The Arduino managed to run the processes with a 1.4 W usage whereas the Pi ran the algorithm with a 3.1 W usage.

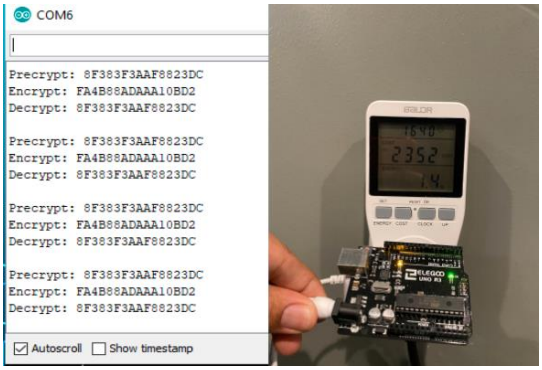

Fig. 4. Result from Arduino process

Fig. 5. Result from Raspberry Pi process

Overall, in terms of simplicity as well as power efficiency it seems obvious that running a full suite basically-microcomputer for a single process vs a microcontroller dedicated to said single process would not be a fair comparison. Regardless, it seems essential to know the differences between the two in terms of efficiency and power usage as well as what the purposes of each are.

V. CONCLUSION

The purpose of this project is to understand performance and information from the metrics of our experiment. The project utilized an Arduino and Raspberry Pi 3 B+ to observe the power efficiency of the XTEA algorithm. Our results indicate that the Arduino is more power efficient than the Raspberry Pi. Although our experiment and results are centered around one metric, it is a step to better understand how to devise different experiments to achieve more complex and informative metrics. An example would be to use the

XTEA algorithm to encrypt/decrypt files of varying sizes and observe the amount of energy used to do so. Performance is essential to determine so that improvements can be implemented to better conserve and utilize resources, whether it be hardware or software related.

REFERENCES

[1] https://en.wikipedia.org/wiki/XTEA

[2] https://en.wikipedia.org/wiki/Tiny_Encryption_Algorithm

[3] https://en.wikipedia.org/wiki/XXTEA

[4] https://cryptography.fandom.com/wiki/XTEAWilliam J Buchanan, *XTEA (eXtended TEA), 2021* https://asecuritysite.com/encryption/xtea

[5] Odzhan, XTEA Block Ciper, January 24, 2018 https://tinycrypt.wordpress.com/2018/01/24/xtea-block-cipher/

[6] Jens-Peter Kaps, Chai-Tea, Cryptographic Harddware Implementation of XTEA, 2008 https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.178.6719&rep=rep1&type=pdf