Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 00 (2019) 000–000

Procedia
Computer Science

www.elsevier.com/locate/procedia

The 19th International Conference on Mobile Systems and Pervasive Computing (MobiSPC)
August 9-11, 2022, Niagara Falls, Canada

New and Efficient Lightweight Cryptography Algorithm for Mobile and Web Applications

Lo'ai Tawalbeh*, Michael Alicea, Izzat Alsmadi

Department of Computing and Cyber Security, Texas A&M University-San Antonio, San Antonio 78224, United States

Abstract

Essentially all forms of online communication involve the use of cryptography to ensure that only the intended recipient of a message can read it. Nearly every protocol includes message encryption as a part of its standard. Whether it is https, sh, or S[MIME, the most common types of connections have encryption enabled by default. Most computers can encrypt and decrypt information without any perceivable difference to the user. Processors found in home computers are fairly powerful, but home computers are not the only devices people have. There are an increasing number of smart devices in your average person's home. These smart IoT devices do not have as powerful hardware as the average PC. To maintain the expected instant transmission of data while providing security, lightweight crypto algorithms are needed for these devices.

Keywords: Mobile Applications Security; Lightweight Cryptography; Authentication; Privacy.

1. Introduction

The transition into the digital era in the 90's and the early 00's was defined by the addition of personal computers into the homes of regular families. For many people, this would be their first machine that ever had an internet connection. Back then, internet speeds were slow. Downloading or uploading nearly anything could be a several hour event. Having multiple internet connected devices could make these already unbearably slow speeds even worse. Over time, internet speeds and latency have drastically improved, nowadays it is no issue to have multiple devices on the same network. Eventually, your average home network consisted of a home computer, maybe a laptop and a few smartphones. We are currently going through yet another change in what your average home network looks like. All

* Corresponding author. Tel.: +1-210-784-5000; fax: +0-000-000-0000.
E-mail address: ltawalbeh@tamusa.edu

sorts of regular appliances are being given networking features, such as light bulbs, refrigerators, TVs, door locks, garage door openers, IP cameras, and in some bored programmers home somewhere, a toaster. It was estimated that in 2020 there was around 50 billion internet connected devices, many of which are IoT devices [4].

IoT devices have a fundamentally different purpose than personal computers (PC). PCs have much of a persons personal information and often transmit that data. Things like passwords, confidential documents, scans of IDs and many other pieces of information get sent from people's PCs. It is important that most of this data remains confidential. This is why some form of encryption is standard in almost every protocol that you would use from your PC including but not limited to https, ssh, S/MIME etc. It is uncommon for data to be sent in plain text from your PC. While passwords and documents are usually not being sent on IoT devices, there is still personal data being transmitted. IP cameras in and around the house, various sensors, whether your doors are locked, all very important information that one would not want a malicious actor to know. This means that encryption is equally important for all of those small bits of data that the various IOT devices send as well. IoT devices are generally very low spec devices that are dedicated to performing a small number of tasks. As they are low cost and low power machines, this has implications on their ability to perform encryption on data.

The key drawback to using normal crypto algorithms on constrained environments such as IoT devices and mobile devices is the limited capabilities such as processing power and storage. To a home PC, algorithms like RSA are no problem and encryption and decryption can happen with no visible delay. While for constrained environments, this will cause a huge delay. With this limitation in hardware/OS capabilities, there is an imposed need for a new category of algorithms to give strong security while being as seamless as PCs. The answer to this problem is lightweight crypto.

Lightweight crypto algorithms seek to give low powered devices a way to perform encryption by having the least amount of computations needed to still provide security while not overburdening the device [3]. One of the ways lightweight crypto algorithms are developed is by taking an existing algorithm and creating a lightweight version of it. This is no new concept, in 2007 DES Lightweight extension (DESL) was introduced as a lightweight version of DES [6]. The concept was simple, take DES and lighten it enough so that low powered devices can perform encryption at a low cost. There are algorithms made with the intention of being used for lightweight crypto. A few of the algorithms we will look at such as PRESENT were created to be lightweight crypto algorithms. Regardless of how the algorithm is made, they share a common end goal, run fast on low power devices.

In this paper, we will identify several lightweight crypto algorithms and their features. We will combine several of these feature in a unique way to create a new lightweight crypto algorithm, Mypher. We will test these algorithms for their performance on resource constrained devices and explore the reasons for their relative performance.

2. Literature Review

While there are many lightweight crypto algorithms, we will focus on the few that will be used as a part of our testing for this project. The three we will focus on are PRINCE, PRESENT and TWINE.

2.1. PRINCE

The PRINCE block cipher is a lightweight crypto algorithm introduced in 2012. The Prince cipher uses a 128 bit key and 64 bit blocks. The first thing PRINCE does is key expansion to 192 bits. This is done by splitting the key in half into two 64 bit keys k0 and k1. The half of the key denoted as k1 is what is used during the cipher rounds. Next a k'0 is created by taking an XOR of the value of a cyclic shift of k0 by the value of a bit-wise shift of k0. The complete 192 bit expanded key is the concatenation of k0, k'0 and k1 [7].

After key expansion, PRINCE moves on to its 12 round process of substitutions and permutations. Each phase consists of the following steps:

- The cipher uses an S-Box for substitutions each round. This S-Box substitutes 4 bits for 4 bits.
- The cipher has 12 predefined round constants known as RC0 through RC11. These constants are iterated through so that each one gets used during their associated round. The current state during that round is XORed with the round constant for that round.

- There is a linear layer in which the current state is multiplied by the matrix M. This matrix is the state with a AES style shift rows applied to it. The exception is the middle round in which the state is multiplied by M1

The PRINCE cipher seeks to give relatively strong encryption at a low cost allowing weaker devices to still enjoy having a moderate level of security.

2.2. PRESENT

The PRESENT cipher is yet another lightweight crypto algorithm meant for resource constrained devices. It is a 64 bit block cipher that supports both 80 bit and 128 bit keys. Unlike PRINCE, there is no key expansion, it is used as is. The authors of the algorithm recommend the use of an 80 bit key for use cases that do not require a high degree of security [1]. PRESENT seeks to be ultra lightweight opting to give enough strength for low a security use case.

The PRESENT cipher uses the 31 round process as follows:

- The PRESENT cipher makes use of a 4 bit to 4 bit S-Box to substitute bits in both the round key and the state of every round.
- Every round of the process involves an XOR with a portion of the key and the current state of the data. Rather than using the entire key each round, the 64 bits on the left side of the key are used in the process of XORing the data. At the end of each round, the key is shifted 61 to the left and then the first 4 bits on the left of the key are run through the S-Box.
- The last step of each of the 31 rounds is to run the data through a permutation box or a P-Box. This P-Box is used to shuffle the bits of the block each round. Unlike PRINCE which performs a cyclic shift, the data is completely scrambled with no real pattern.

PRESENT is fairly low cost, even amongst other low cost algorithms. If one opts into using an 80 bit key, it is less secure. The best use case would be for when a large number of small transactions need to occur in rapid succession in which it is not critical if any individual transmission is cracked.

2.3. TWINE

Twine is yet another light eight crypto algorithm and was introduced in 2012. Similarly to PRESENT, TWINE is a 64 bit block cipher that supports both 80 bit and 128 bit key sized [9]. TWINE was created for lightweight devices going as low as RFID chips up to micro-controllers with Intel Core-i series processors.

The TWINE cipher has a 36 round process with the following features:

- Similarly to the other ciphers, TWINE makes use of a 4 bit to 4 bit S-Box.
- The key schedule contains two parts. The first is that the S-Box is applied to the round key so that the key changes between rounds. Secondly, a round constant is used to perform a left cyclic shift on the key.
- Similarly to PRESENT, TWINE elects to not perform cyclic shift on the data on rounds. Instead the state of the data in each round is run through what they call a block shuffle

TWINE is a little different as the authors sought to make the algorithm multi-platform, running efficiently on both resource constrained devices like RFID chips and Intel home CPUs as well [8].

3. Mypher

Using a combination of the features from the ciphers above, we created our own cipher. Using the same conventions as the ciphers from the previous section, Mypher is a 64 bit block cipher with an 80 bit key. This cipher is a lightweight crypto algorithm seeking to provide moderate security at a low cost. It would ideally be used on IOT devices sending small amounts of data at a time, such as from sensors. The implementation created for this paper was written in the Python programming language.

The following sections will explain the use and the order of the features in the algorithm during its encryption and decryption processes.

3.1. Substitution Box

The algorithm makes use of a 4 bit to 4 bit substitution box. This box is used to change the values of the 4 bit blocks during the encryption cycles. The substitution box is also used during the key cycle to change the bits of the keys as shown below:

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
D	9	5	F	0	3	1	C	2	4	B	E	7	A	6	8

Table 1. Mypher Substitution Box

This substitution box is used every encryption round to change the data, and change the key. This changes both parts of the XOR step so that it adds difficulty to crack.

3.2. Permutation Box

Our algorithm makes use of a permutation box. While a permutation box is similar in appearance to a substitution box, it has a different function. Rather than changing the values of the data, it changes the position. The permutation box serves to shuffle the data during the encryption steps. Unlike a substitution box however, the permutation box can not be used on the key. The key is a different size than the block size and therefore is too long to run through the permutation box as described in the below Table:

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
9	C	5	0	F	B	1	A	E	3	6	2	8	7	D	4

Table 2. Mypher Permutation Box

3.3. Key Cycle

At the beginning of the encryption process, the user supplies the algorithm with their 80 bit key. A portion of this key is used during the XOR step in the process. As the key is longer than the block size, only the first 64 bits of the key are used during each XOR stage. This means that we use a portion of the key that is equal in size to that of the block. During each of the encryption rounds, there are two alterations made to the key.

- The first is a cyclic key shift. Between every round, the key is shifted 16 bits to the right. What this accomplishes is changing which portion of the key is used during each round.
- The second modification is that the key is run through the substitution box. Unlike the permutation box, the substitution box does not require the data run through it to be a set size. The 4 bit substitution takes place over the entire key after the keyshift has occurred.

By altering the key between each of the encryption rounds, we can ensure that neither of the two values with which the XOR is performed at any stage are the same value as a previous round.

3.4. The Encryption and Decryption Process

Know that we have explained the features of the algorithm and how they are used during the encryption and decryption process, we can now discuss how each of these pieces fit together to create Mypher.

- The first step in the encryption process is that both the data and the key are converted into hex.

- After the data and key have been converted we begin the 75 round process of encrypting a 64 bit block. The first step in each round is the initial XOR of the data with the first 64 bits of the key. This represents the very first change to the data.
- After the XOR has been performed on the data, it is run through the substitution box. It substitutes 4 bits for 4 bits. This effectively changes the data before going into the next round.
- After the data has been altered with substitution, it is then run through the permutation box. After this step, the data is effectively shuffled for the next encryption round.
- Before an encryption round ends, the key cycle modifications are made. First the key is shifted 16 bits to the right. After the keyshift, the key is ran through the substitution box to change the key between rounds.
- This process is then repeated. In total, the algorithm runs for 75 rounds.

The decryption process is nearly identical to the reverse process. There is a slight difference as the key has to be in the state that it would be in at the end of the encryption process. By the end of the 75 encryption rounds, the key is in a different state as it has been shifted and run through the substitution box 75 times. Due to this, decryption has the additional step of running the key through its encryption key cycle 75 times.

After the key is in the same state as the end of an encryption, the encryption process is followed in reverse to decrypt. The XOR is identical as the process can be reversed simply by performing another XOR with the key on the resulting cipher text. As for the substitution and the permutation, two special functions have been made to undo these two processes. The reverse substitution makes use of a substitution box that is the inverse of the encryption substitution box. The data is ran through the two functions that reverse the effects of the permutation and substitution. Similar to the data, the key is also put through the function to undo the substitution process and is then shifted 16 bits to the left. This process also takes 75 rounds and results in the initial plain text.

4. Testing The Algorithms

The proposed algorithm will be put through a test along with the three algorithms listed in the literature review, PRINCE, TWINE and PRESENT. Lightweight crypto algorithms exist for the purpose of being faster than normal crypto algorithms due to the constrained nature of the devices that they are intended to run on. Due to this, the speed of these algorithms will be the primary focus of the test that will be outlined.

4.1. Environment

Given that the purpose of lightweight crypto algorithms is to run quickly on resource constrained IOT devices, testing them should best represent the use case that it would most likely have. Unfortunately, many commercial IOT devices are proprietary devices that do not allow for much tampering, so attempting a test on these devices is not feasible. There are however, low powered devices that are commercially available that are more open in nature.

The Raspberry Pi is a good fit for this type of test, as it is not only an open platform to develop on, but it is also quite accessible. The Raspberry Pi is a credit card sized computer running on an ARM processor. These devices make ideal test beds due to their relatively low power processors making it a good representation of a constrained environment such as IoT devices and Mobile computing and applications. For the following test, we will be using two different models of Raspberry Pi:

The first is a Raspberry Pi 3B+. This computer has an ARMv8, quad core processor clocked at 1.4Ghz. This is accompanied with 1GB of ram. [2]. The second is a Raspberry Pi 4B. This model has an ARMv8, quad core processor clocked at 1.5Ghz. The particular model we have has 4GB of RAM [5].

Both of these devices will be running the newest version of Raspberry Pi OS.

Another test will be conducted using a AMD64 CPU. This test will be performed on a laptop with an Intel Core i7-8750H with 8GB of ram. The purpose of this final test is to give a point of reference to personal computers. Testing on a PC helps demonstrate the need for lightweight crypto algorithms. The laptop will excel by comparison, but it was designed to have plentiful resources for the user. In constrained environments the resources perform particular task with minimal wasted potential such as unused processing cores or empty ram.

4.2. Programming Language and Test Specifications

Different programming languages have different performance. Compiled languages tend to be faster than interpreted ones. The same algorithm for example in the compiled language C, may have very different performance than an interpreted language such as Python. Even among the same category of compiled or interpreted can result in different levels of speed. To ensure that the algorithms have the most fair comparison, all algorithms will use the same programming language, Python. Every one of these algorithms have a Python implementation, and Python was the language of choice for our algorithm.

As it is an interpreted language, Python is highly portable. The algorithms can be developed on an AMD64 computer, and the code can still run on ARM. There is the added benefit that interpreted languages can use the same code across different operating systems, so long as no special features to the operating system are used, which none of the chosen algorithms do. This allows for the testing of these algorithms to be easily expanded upon in the future. With ARM computers becoming an option even for Windows and Mac, these tests can run on those operating systems without any modification of the code.

In order to test the various encryption algorithms against one another, the tests will be as uniform as possible in order to give the best results.

4.2.1. Key Size

As the test is for lightweight cryptography applications, the test will opt to use the smallest key size available for each of the algorithms. For TWINE, PRESENT and Mypher, this is an 80 bit key. The only exception for this is PRINCE, which only accepts 128 bit keys. This will result in PRINCE having drastically slower speeds, but PRINCE will provide more security. It is also important to note that not all IoT devices are created equal, there are some IoT devices that do have respectable computing power. Regardless, this test will result in PRINCE being less comparable, but still a useful data point.

4.2.2. Block Size

Every one of the lightweight crypto algorithms are 64 bit block ciphers. This means that each one of these algorithms can encrypt 8 bytes, or 8 ASCII characters at a time. Since we want to time each of these algorithms for their speed, rather than their python implementations ability to queue multiple blocks, just a single block will be used. Before the algorithms are tested, 1000 strings of 8 random printable ASCII characters will be generated. Performing the test this way ensures that there is no interference in either queuing up the next block and the issue of padding up a block that is less than 64 bits.

4.3. Performance Tests

As per the specifications listed in the previous section, the test of each of these algorithms will be a series of timed encrypts and decrypts. The previous section lists how these encrypts and decrypts will be kept as uniform as possible to ensure that each of the algorithms can be tested for their speed on low powered devices like the Raspberry Pis 3 and 4. The encryption test will be performed as:

- A list of 1000, 8 character strings will be generated before any testing begins. As each algorithm is a 64 bit block cipher, 8 characters create a single block. Each algorithm will use this same list of strings.
- With the exception of PRINCE, every algorithm will be supplied the same key. Each algorithm uses an 80 bit key, therefore to keep things as equal as possible, the same 10 character key will be used for each algorithm.
- For each algorithm, a timed loop will be used to encrypt all 1000 strings. The results of each encryption will be stored into a separate list for the decryption test later.
- The reported time for encryption will be the result of how long each algorithm took to encrypt all 1000 strings.

The decryption test will be much the same. As each of the strings was encrypted, the same key must be used to decrypt. The decryption test will consist of: - The initial list of 1000 random strings is now encrypted. This new list of encrypted strings will be used for decryption. - A timed loop will be used for each algorithm to decrypt the strings that they had just encrypted. - The reported time for decryption will be the result of how long each algorithm took to decrypt all 1000 strings. This test should provide comparative results of the speed of each algorithm. By using 1000 encryptions and decryptions, we can limit the affect of any performance variation with the volume of operations to give a clear average of how long each algorithm takes to perform its operations.

5. Performance Experimental Results

Each of the tests were run on the three machines. The times shown are the time taken to perform 1000 encrypts and 1000 decrypts displayed in seconds. The following figures display the results of those tests.

Raspberry Pi 3	Encryption	Decryption
Mypher	4.17118s	5.32321s
Twine	4.154831s	6.44118s
PRINCE	156.96444s	157.22997s
PRESENT	5.79016s	5.78759s
Raspberry Pi 4	Encryption	Decryption
Mypher	1.45062s	1.83058s
Twine	1.50299s	2.38576s
PRINCE	71.47251s	71.65946s
PRESENT	2.03991s	2.02709s
Core i7	Encryption	Decryption
Mypher	0.29653s	0.37770s
Twine	0.27602s	0.42392s
PRINCE	13.68296s	13.65142s
PRESENT	0.37993s	0.39156s

Table 3. The Performance Tests results Using Raspberry Pi 3, Raspberry Pi 4, and Core i7

From these experimental results, we can notice that PRINCE was the slowest algorithm by a large margin. This is due to it having the most complicated cycle alongside the largest minimum key. If PRINCE is being used to encrypt large amounts of data, it will be a bottleneck. This test is however not meant to represent a real world application but rather to display performance. IoT devices typically send out small transactions on a regular basis rather than a large transaction all at once. PRINCE is not necessarily unusable due to its slower speed. In fact this can still be a benefit to it. PRINCE offers the best security which makes it suitable to protect sensitive data in at slower intervals .

Next is the relative performance of the algorithms. PRINCE aside, the algorithms all have comparative performance with PRESENT being approximately 36% slower than Mypher and Twine. These algorithms provide lesser security to PRINCE due to their less complex encryption rounds, however they are faster. PRESENT, Twine and Mypher are better suited to applications where the data is not incredibly sensitive but one would not want an attacker to be able to crack a large amount of data. These three algorithms are better suited for applications when data is sent often, continuous even. The more transactions there are, the better the value proposition of these algorithms. They are fast and can provide a moderate amount of security.

Another trend that was noticed in the algorithms is that decryption times are consistently slower than encryption times with the exception of PRESENT in which times were so close that different runs had different results. As the decryption process in theory is simply the encryption process in reverse, both processes should have roughly the same cost and therefore the same performance. In reality this is not the case. There is one piece of the decryption process in which the majority of this slowdown occurs. The decryption process for each of these algorithms has an extra step. Among this group of algorithms, the key itself gets changed throughout the encryption process. The key is often shifted and run through the substitution box. While it is trivial to determine what the key would look like after X amount of cyclic shifts, the same is not true for substitution. There is no easy way to take a substitution box and calculate what any given 4 bits should be after X amount of substitutions, especially when bits are being shifted around. The easiest way to determine what the key should look like after the shifts and substitutions is to perform the operations. This results in the decryption algorithm running X amount of times to determine the key at the end of the encryption process, then running X rounds again to decrypt. It is this accounting for the key cycle that adds time to decryption that the encryption process does not need to account for.

Also, we can notice there is a difference in performance between the Raspberry Pi 3 and the Raspberry Pi 4. While these two devices are resource constrained devices, the Raspberry Pi 4 was more than twice as fast in each of the events than the Raspberry Pi 3. Every one of the algorithms tested used a single threaded process, so there was no issues regarding the usage of multiple threads to speed up the process. This was the difference between the two processors on each machine. It was expected that the Raspberry Pi 4 would be faster, but it was not expected to be that much faster. This shows that in constrained environments, one specific algorithm may not be suitable for data that is equally as sensitive. As each device has differing amounts of resources, there is no easy way to definitively say that one algorithm is better for a specific use case. Picking the right lightweight crypto algorithm will take testing to ensure that the performance to security equation gives an acceptable level of risk for the chosen hardware.

6. Conclusion

This paper laid out the various features used by a select number of lightweight crypto algorithms. We explored how each algorithm used the various techniques in order to provide security to IoT devices that have constrained resources. Through the use of lightweight crypto algorithms, IoT devices are capable of providing some level of security, without having the massive impacts to performance that normal algorithms would have. Rather than providing top of the line security, these algorithms seek to provide moderate security at the lowest possible cost. By reviewing these techniques, we were able to create a new lightweight crypto algorithm. By combining the usage of a substitution box, a permutation box and cyclic key shifting, the algorithm Mypher was created.

Through the use of python implementations of each of the algorithms, the relative performance of each algorithm was obtained. It was found that Mypher had a similar performance across both the Raspberry Pi 3 and 4 compared to the existing algorithms. Through this testing and through comparison of the code, we were able to identify key aspects as to why algorithms performed the way that they did and differences between the encryption and decryption process. It was also identified that even among IoT devices, there can be widely varying performance. This shows that there is no one size fits all solution to picking an algorithm even amongst resource constrained devices.

Acknowledgements This material is based upon work supported by the National Science Foundation (NSF) under Grant No. 2131193. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J., Seurin, Y., Vikkelsoe, C., 2007. Present: An ultra-lightweight block cipher, in: International workshop on cryptographic hardware and embedded systems, Springer. pp. 450–466.

[2] Hattersley, L., is Editor of The MagPi, L., . Raspberry pi 3b+ specs and benchmarks. URL: https://magpi.raspberrypi.com/articles/raspberry-pi-3bplus-specs-benchmarks.

[3] Lo'ai, A.T., Tawalbeh, H., 2017. Lightweight crypto and security. Security and Privacy in Cyber Physical Systems: Foundations, Principles and Applications , 243–261.

[4] Meneghello, F., Calore, M., Zucchetto, D., Polese, M., Zanella, A., 2019. Iot: Internet of threats? a survey of practical security vulnerabilities in real iot devices. IEEE Internet of Things Journal 6, 8182–8201.

[5] Pi, R., . Buy a raspberry pi 4 model b. URL: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/.

[6] Poschmann, A., Leander, G., Schramm, K., Paar, C., 2007. New light-weight crypto algorithms for rfid, in: 2007 IEEE International Symposium on Circuits and Systems, IEEE. pp. 1843–1846.

[7] Rashidi, B., 2020. Low-cost and two-cycle hardware structures of prince lightweight block cipher. International Journal of Circuit Theory and Applications 48, 1227–1243. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/cta.2832, doi:https://doi.org/10.1002/cta.2832, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/cta.2832.

[8] Suzaki, T., Minematsu, K., Morioka, S., Kobayashi, E., 2012. Twine: A lightweight block cipher for multiple platforms, in: International Conference on Selected Areas in Cryptography, Springer. pp. 339–354.

[9] Wei, Y., Xu, P., Rong, Y., 2019. Related-key impossible differential cryptanalysis on lightweight cipher twine. Journal of Ambient Intelligence and Humanized Computing 10, 509–517.