An Encryption Scheme using Dynamic Keys and Stream Ciphers for Embedded Devices

Chrysoula Oikonomou
Information Technologies Institute
Thessaloniki, Greece
chrisaoikon@iti.gr

Charalampos S Kouzinopoulos
Information Technologies Institute
Thessaloniki, Greece
kouzinopoulos@iti.gr

Dimosthenis Ioannidis
Information Technologies Institute
Thessaloniki, Greece
djoannid@iti.gr

Dimitrios Tzovaras
Information Technologies Institute
Thessaloniki, Greece
Dimitrios.Tzovaras@iti.gr

***Abstract*—Security in embedded devices can be challenging, due to limited available resources, including processing power, memory and energy autonomy. Embedded security solutions should have a low computational complexity and minimum memory requirements, while at the same time these must not reflect in the mechanism's efficiency. This paper describes a lightweight scheme for Data in Transit Encryption (DiTE) using dynamic keys, designed for embedded devices. The scheme is based on the RC4 and ChaCha ciphers and uses communication channel characteristics, and especially the RSSi, to generate the encryption keys. The proposed algorithm provides an enhanced security level, due to the non static encryption keys, in both physical and transport level.**

***Index Terms*—cybersecurity, encryption, embedded, lightweight, dynamic key**

I. Introduction

The number of smart devices that are connected through the Internet has significantly increased over the past decade, due to the emergence of the IoT paradigm. According to Cisco [1], IoT devices will account for 50% (14.7 billion) of all global networked devices by 2023. Edge nodes and embedded devices in IoT are often targeted by malicious users, as they may constitute an intermediate node to a wider system, or be a source of valuable information. Hence, protecting the information exchanged between embedded devices is a crucial step towards privacy preservation and system security.

There is already a variety of encryption algorithms available, however there is significant research on the design of lightweight algorithms that utilise efficiently the restrained capabilities of embedded devices. This paper introduces a lightweight encryption scheme for embedded systems that combines dynamic key generation, based on the physical layer characteristics, in combination with a stream cipher. The scheme is evaluated with four different stream cipher encryption algorithms, to determine the optimal one in terms of energy consumption and latency induced. The proposed encryption scheme provides enhanced security due to the key's dynamic nature. The scheme has been tested on an autonomous embedded system for environmental sensing and asset tracking in smart IoT environments. The implementation evaluation showed low RAM requirements.

The rest of the paper is arranged as follows: Section II describes related work on encryption algorithms for embedded devices. In Section III, a detailed analysis of the proposed encryption scheme is given, while Section IV discusses the cost of the schema implementation, in terms of energy consumption and latency. Finally, the conclusions of this work as well as discussion on future work are presented in Section V.

II. Related Work

Research in the design of lightweight encryption algorithms generally focuses on the reduction of their computational cost while also maintaining a high level of security. ALE [2] is an online, single-pass, nonce-based authenticated encryption technique with associated data support as an option. A nonce is defined as a random or pseudo-random number that is required by the encryption process, that can only be used once. Plain-text lengths of up to 245 bytes are supported. Authenticated Stream-Cipher (ASC) [3] has a similar overall structure to ALE. The AES-NI assembly instruction set is used due to its improved security and performance.

JAMBU, a lightweight block cipher based on the AES-128 algorithm and the SIMON block cipher, is introduced in [4]. The cipher, that uses a nonce-based key and XOR operations, is efficient in terms of power consumption. If there is any misuse of nonce, namely in the case it is used multiple times, the Ciphertext feedback (CFB) [5] algorithm can be used. CFB can be implemented in devices with limited processing power and memory capabilities, including RFID devices. It is based on the Compact Low-Overhead CFB (CLOC) algorithm [6] but with a reduced computational complexity.

A cipher scheme is presented in [7], based on Physical Layer Security (PLS). PLS schemes use the temporary and random characteristics of a wireless communication channel to transform the encryption into a dynamic process. Namely, the generation of the key used for encryption in such schemes is directly linked to temporary channel characteristics, and thus it does not stay static over time. At the same time, no assumption needs to be made for the end devices, as the physical layer is shared to any kind of device. Therefore, such schemes are suitable for any device with wireless communication capabilities. The scheme includes a secret key which is calculated by both endpoints of a communication link, using an estimation of the Channel State Information (CSI). The drawback of this algorithm lays on the fact that CSI information can also be obtained by a malicious user that can potentially estimate the channel properties of the communication. This was addressed in [8], by encrypting the preamble of network packets. The preamble has a length of 56 bits and allows communicating devices to synchronise their receiver clocks, providing bit-level synchronisation.

III. Proposed DiTE Design

A. Stream Ciphers and Reused Key attacks

Stream Ciphers are encryption algorithms used to encrypt data by combining it with a keystream. During the encryption process, the plaintext is being processed on a digit by digit basis, along with the corresponding digits of the keystream, in order to produce the encrypted sequence. To combine the keystream and the plaintext data, an XOR operation is frequently used. Stream ciphers are more efficient than other types of ciphers, such as block ciphers, in terms of execution time and hardware complexity, however they are vulnerable to attacks known as Reused Key attacks. Such attacks are based in the use of the same encryption key more than once. In that case, an attacker may retrieve the plaintext information due to the commutative property of the XOR operation. In order to address this threat, an input that can not be predicted due to its (pseudo)random nature is added to provide the initial state of the cipher, known as Initial Vector (IV).

The stream ciphers evaluated in the context of this paper's implementation are the ChaCha [11] HC-128 [12], Rabbit [13] and RC4 [14] ciphers. The selection of these algorithms was based on their minimal requirements in terms of memory and processing power. All of the above work by generating lengthy key-stream sequences and appending them to data bytes. The ciphers encrypt data by XORing it sequentially with key-stream bytes. The first three algorithms use an IV during their initialisation phase while the RC4 does not. Instead, the static nature of the RC4 secret key is transformed by introducing a nonce in the key generation process, derived by physical channel characteristics. The rest of the algorithms are enhanced, in terms of security, by the added nonce.

B. DiTE Scheme

The DiTE scheme is used to encrypt network packets. The process begins after the message to be sent is formed into a structured network packet, the input frame. Each input frame is divided into three parts:

- Public information
- Frame preamble
- Actual data frame

The first part is kept in plain-text while the second and third parts are encrypted in phases $L2$ and $L3$ of the scheme. A visualisation of the DiTE scheme is presented in Fig. 1

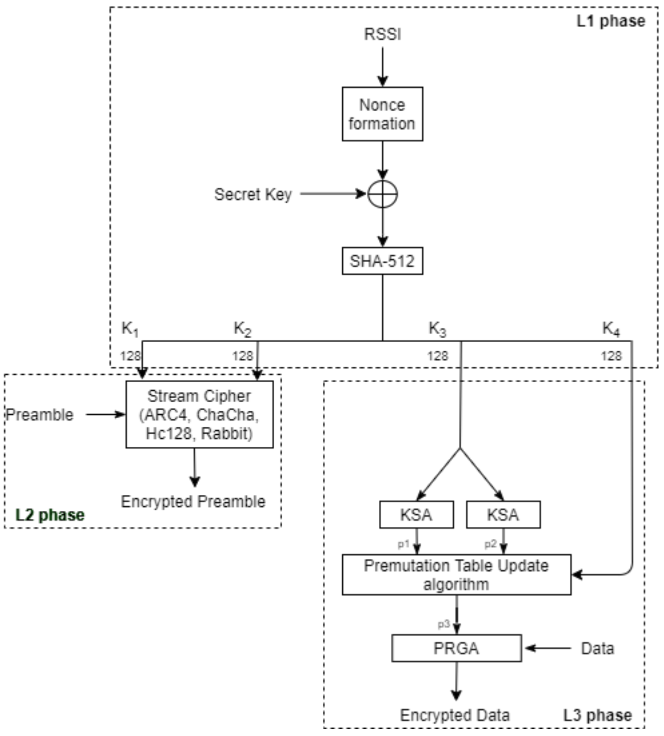

Fig. 1. Key Generation and Encryption Scheme

The proposed scheme has three phases of execution. The first one, $L1$, is the key generation process, which requires a secret key and a nonce. The secret key is stored encrypted in the memory of the low-power platform and is shared among the trusted devices of the network, while the nonce is produced using the channel characteristics. It is important to note that the same nonce is required by both ends of the communication, so a high level of synchronisation is required between them to produce the exact same nonce. For the nonce generation, the Received signal strength indication (RSSi) is used, which measures the power in a received radio signal. Initially, RSSi is transformed into a bit-stream and is subsequently hashed. After obtaining both the secret key and the nonce, a XOR operation is performed. Then, the result of the operation is hashed. For the hash function, the SHA-512 key is used, therefore the hashed value has a length of 512 bits. This is the dynamic key. The dynamic key is then split into four parts of equal length ($K1$, $K2$, $K3$, $K4$), with each part consisting of 128 bits.

However, if a key is solely derived from the communication channel characteristics, it can be possible for an attacker to calculate it. Thus, a lot of attention needs to be paid during the key generation process to maintain the randomness of the channel characteristics and add a certain complexity to the formation of the keys. A further security measure that was taken in order to address this weakness is the preamble encryption. The preamble is utilised in the receiver to accomplish frequency synchronisation, automatic gain control (AGC) training, and symbol timing estimation. Thus, if such information is available in plain text, attackers may be able to retrieve channel characteristics that can lead to the encryption key. The preamble encryption, or $L2$ phase, uses $K1$ and $K2$ as the required

TABLE I
LATENCY INDUCED FROM THE STREAM CIPHERS

	Key Generation	Encryption
ChaCha	512 μs	608 μs
HC-128	19712 μs	323 μs
RC4	1111 μs	190 μs
Rabbit	1763 μs	86 μs

```
Key-Scheduling Algorithm
for i ← 0 to N do
    S[i] ← i
end for
j ← 0
for i ←0 to N do
    j ← (j +S[i] + k[j mod keylen] mod N)
    swap (S[i], S[j])
end for
return S
```

Fig. 2. RC4 KSA

seed and IV to a stream cipher that encrypts the preamble. As previously mentioned, three out of the four algorithms tested, use both a seed and an IV. In these cases, $K1$ is used as the seed-key while $K2$ is used as the IV. In the case of the RC4 cipher, $K1$ and $K2$ are concatenated to form the single key required. The plaintext preamble is then encrypted with this cipher.

Phase $L3$ includes the key generation process and the encryption of the actual data needed to be transmitted, using a variation of the RC4 algorithm. Sub-keys $K3$ and $K4$ are utilised for the formation of the permutation tables $p1$ and $p2$, that are used to generate the final permutation table $p3$. the algorithm used to generate the permutation tables is the Key-Scheduling Algorithm (KSA) of the RC4 stream cipher, presented in Fig. 2. K is the secret key while N is the length of the permutation table. $p3$ is generated using the algorithm shown in Fig. 4. This algorithm also requires the use of a byte-stream, that in the current context corresponds to 4. Finally, $p3$ is used to generate the keystream that is XORed with the actual data using the Pseudo-Random Generator Algorithm (PRGA) of RC4, as given in Fig. 3.

IV. IMPLEMENTATION AND RESOURCES REQUIREMENTS

The proposed encryption scheme of this paper is based on the work presented in [9]. This paper presents an implementation of the aforementioned work on a low-power platform and suggests the use of the ChaCha stream cipher in the $L2$ phase of the scheme. The platform used was the following characteristics. The MCU of the system is based on the high-performance ARM Cortex-M4 32-bit RISC core operating at a frequency of up to 80MHz. The core features a floating-point single-precision unit which supports all ARM single-precision data-processing instructions and data types. It also has 1MB of Flash memory and 320KB of SRAM.

```
Pseudo-Random Generation Algorithm
i ←0
j ←0
for r←1 to N do
    i ← (i+1) % N
    j ← (j+S[i]) % N
    swap (S[i], S[j])
    cipher ← S[ (S[i] + S[j]) % N ]
    ciphertext[n] ← cipher ⊕ plaintext[n]
end for
return ciphertext
```

Fig. 3. RC4 PRGA

```
Permutation Update Algorithm
for i ← 0 to N do
    p3 [i] ← i
end for
for i ← 0 to N do
    if hash[i] > 0x80 then
        p3 [i] = p1 [i]
    else
        p3 [i] = p2 [i]
    end if
end for
return p3
```

Fig. 4. Permutation Table Update Algorithm

The procedure followed for both encryption and decryption is similar. The first step includes the keys generation for both the receiver and the transmitter. The information required during this process is available to both of them, without the need to exchange further data. The challenging point in this step implementation was the RSSi calculation. The RSSi is susceptible to multiple characteristics of the endpoints as well as the environment. In order to maintain the RSSi stable during the implementation, the following measures were taken to ensure that there were no interference from the environment:

- No physical obstacles were present between the two communicating devices
- The distance between the two communicating devices was constant
- There were no other 802.11 sources that might cause radio interference

After the keys have been obtained, a XOR operation is performed for the encryption or the decryption of the plain/cipher text. Hence the requirements are common for the transmitter and the receiver.

Table I presents the latency that is induced by each of the stream ciphers, used the for the preamble encryption. In order to calculate the energy required for the code execution, specialised hardware that enables the accurate calculation of the system power consumption was used. During the execution time, a total of 4.37KB RAM is used and 31.23KB of flash memory.

As far as the $L1$ phase is concerned, power consumption and latency is the same for all cases and equal to $3.36\mu J$ and $6458\mu s$. The same is true for the $L3$ phase as well. The power consumption for the key generation and encryption processes in $L3$ are $3.07\mu J$ and $0.06\mu J$, while latency is $5892\mu s$ and $106\mu s$, respectively. Table II lists the power consumption for the key generation and encryption processes executed in the $L2$ phase.

TABLE II
POWER CONSUMPTION OF EACH STREAM CIPHER

	Key Generation	Encryption
ChaCha	0.267 μJ	0.317 μJ
HC-128	10.27 μJ	0.168 μJ
RC4	0.919 μJ	0.0448 μJ
Rabbit	0.579 μJ	0.099 μJ

As can be seen, the HC-128 candidate was the first to be eliminated as it required by far the biggest power consumption in the key generation phase, followed by the RC4. The final choice was based on the key generation requirements, as the unstable nature of the RSSi which acts as the scheme nonce, implies that the key generation process is executed more often than in cases where the key is static. Hence the ChaCha candidate was selected. The table III summarises the total costs of the final proposed scheme.

V. CONCLUSIONS AND FUTURE WORK

The work presented in this paper details the implementation of an encryption scheme, that uses dynamic keys based on the temporary communication channel characteristics, and specifically the RSSi. The obtained RSSi is hashed with a secret key in order to produce sub-keys that are used as input to the ciphers. The scheme includes key generation and encryption processes for both the dataframe and the preamble of the packet. The dataframe encryption is performed using a variation of the RC4 cipher while the preamble encryption, is performed with the ChaCha cipher. The design is energy efficient and it is proposed for energy and memory constrained devices. As a next step, we plan to redesign the scheme using ChaCha and Rabbit variations in the $L3$ encryption, in order to test if it can be further optimised in terms of latency and power consumption.

ACKNOWLEDGEMENT

This work is co-funded by the European Union (EU) within the AMANDA project under grant agreement number 825464. The AMANDA project is part of the EU Framework Programme for Research and Innovation Horizon 2020.

TABLE III
LATENCY AND POWER CONSUMPTION OF THE FINAL SCHEME

	Hashing and Sub-keying	Preamble Encryption	Data Encryption
Latency	6458 μs	512 μs + 608 μs	5892 μs + 106 μs
Power Cons.	3.36 μJ	0.267 μJ + 0.317 μJ	3.07 μJ + 0.06 μJ

REFERENCES

[1] Cisco ”Cisco Annual Internet Report - Cisco Annual Internet Report (2018–2023) White Paper”, Cisco, 2022. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html. [Accessed: 31- Jan- 2022].

[2] Bogdanov A., Mendel F., Regazzoni F., Rijmen V., Tischhauser E. (2014) ALE: AES-Based Lightweight Authenticated Encryption. In: Moriai S. (eds) Fast Software Encryption. FSE 2013. Lecture Notes in Computer Science, vol 8424. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-43933-3_23

[3] G. Jakimoski and S. Khajuria, “ASC-1: an authenticated encryption stream cipher,” in 18th InternationalWorkshop on Selected Areas in Cryptography, Toronto, 2011.

[4] H. Wu and T. Huang, “JAMBU Lightweight Authenticated Encryption Mode and AES-JAMBU,” 2014. [Online]. Available: https://competitions.cr.yp.to/round3/jambuv21.pdf.

[5] T. Iwata, K. Minematsu, J. Guo, S. Morioka and E. Kobayashi, ”SILC: SImple Lightweight CFB,” 2015. [Online]. Available: https://competitions.cr.yp.to/round2/silcv2.pdf.

[6] T. Iwata, K. Minematsu, J. Guo and S. Morioka, ”CLOC: Compact Low-Overhead CFB,” 2014. [Online]. Available: http://competitions.cr.yp.to/round1/clocv1.pdf.

[7] J. M. Hamamreh and H. Arslan, ”Secure Orthogonal Transform Division Multiplexing (OTDM) Waveform for 5G and Beyond,” 2014 IEEE International Conference on Communications Workshops (ICC), pp. 1191-1194, 2017.

[8] H. Rahbari and M. Krunz, ”Exploiting frame preamble waveforms to Support New Physical-Layer Functions in OFDM-Based 802.11 Systems,” EEE Transactions on Wireless Communications, pp. 3775-3786, June 2017.

[9] H. Noura, R. Melki, A. Chehab and M. Mansour, ”A Physical Encryption Scheme for Low-Power Wireless M2M Devices: a Dynamic Key Approach”, Mobile Networks and Applications, vol. 24, no. 2, pp. 447-463, 2018. Available: 10.1007/s11036-018-1151-7.

[10] ”Encyclopedia of Cryptography and Security”, 2011. Available: 10.1007/978-1-4419-5906-5 [Accessed 11 February 2022].

[11] Bernstein, Daniel J. ”ChaCha, a variant of Salsa20.” Workshop record of SASC. Vol. 8. No. 1. 2008.

[12] Wu, Hongjun. ”The stream cipher HC-128.” New stream cipher designs. Springer, Berlin, Heidelberg, 2008. 39-47.

[13] Boesgaard, Martin, et al. ”Rabbit: A new high-performance stream cipher.” International workshop on fast software encryption. Springer, Berlin, Heidelberg, 2003.

[14] Paul, Goutam, and Subhamoy Maitra. RC4 stream cipher and its variants. CRC press, 2011.