# Event-based cryptography for automation networks of cyber-physical systems using the stream cipher ChaCha20

**Públio M. Lima** * **Carlos K. P. da Silva** * **Claudio M. de Farias** **
**Lilian K. Carvalho** * **Marcos V. Moreira** *

* *COPPE - Electrical Engineering Program, Universidade Federal do Rio de Janeiro, Cidade Universitária, Ilha do Fundão, Rio de Janeiro, 21.945-970, RJ, Brazil,*
*Emails: publio@poli.ufrj.br, carloskpereira.07@poli.ufrj.br , lilian.carvalho@poli.ufrj.br, moreira.mv@poli.ufrj.br*
** *COPPE - PESC, Universidade Federal do Rio de Janeiro, Cidade Universitária, Ilha do Fundão, Rio de Janeiro, 21.945-970, RJ, Brazil,*
*Email: cmicelifarias@cos.ufrj.br*

**Abstract:** One of the main concerns about implementing cyber-physical systems (CPS) is ensuring its security against cyber attacks. In this paper, we consider CPS in a Discrete-Event Systems (DES) framework, and consider cyber attacks in the automation network of the CPS, where a malicious agent eavesdrops communication channels with the objective to gather information about the system behavior. It is important to remark that network security strategies used in Information Technology (IT) cannot be straightforwardly used in automation networks, and adding a new layer for security may compromise the transmission time. In this paper, we propose a cryptographic scheme to be applied in an automation network which cipher events without altering the size or structure of the transmitted data. In addition, the proposed cryptographic scheme leads to small communication delays, which makes it suitable for application in automation networks. We call this scheme event-based cryptography, where an event is defined as any change in the binary signals transmitted in the network. In order to do so, we propose a method for the codification of events as event vectors, which is suitable for encryption. We also propose the use of a stream cipher called ChaCha20, which is known to have a high resistance to cryptanalysis. A simulated example is used to illustrate the application of the proposed event-based cryptography, and a comparison with the RSA cipher, a public-key cipher widely used in IT, is presented.

*Keywords:* Cyber-Security, Cryptography, Cyber-Physical Systems, Discrete-Event Systems.

## 1. INTRODUCTION

Modern engineering solutions consider the implementation of cyber-physical systems (CPS), which consist of systems that integrate computing and communication capabilities to monitor and control physical processes (Tao et al., 2019; Lima et al., 2019; Alwan et al., 2022). Since CPS use communication networks, then one of the main concerns about using this type of system is ensuring its security against cyber attacks (Song et al., 2016; Yaacoub et al., 2020; Lima et al., 2021). In this paper, we consider CPS in a Discrete-Event Systems (DES) framework, and consider cyber attacks where a malicious agent eavesdrops the communication channels of the automation network of the CPS, with the objective to gather information about the system behavior.

⋆ This work has been partially supported by the National Council for Scientific and Technological Development - Brasil (CNPq) - under grants 305267/2018-3, 431307/2018-0, and 436672/2018-9, FAPERJ, and the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

In the context of DESs, the security against attacks whose objective is to estimate a secret behavior executed by the system is usually addressed by using obfuscation policies (Lin, 2011; Yin and Lafortune, 2015; Jacob et al., 2016; Tong et al., 2018; Lafortune et al., 2018; Ji et al., 2018; Barcelos and Basilio, 2021; Li et al., 2021; Basilio et al., 2021). Another way of protecting data in communication channels is the use of cryptography (Stallings, 2006; Kurose and Ross, 2011; Fritz and Zhang, 2018; Fritz et al., 2019; Lima et al., 2020). In Fritz et al. (2019), the authors propose the use of cryptography based on a fully homomorphic encryption scheme (Gentry, 2009) in a networked automation system composed of a plant and a controller, modeled as DES. A controller encryption scheme is proposed to secure the communication and the information inside the controller. Since the encryption method presented in Fritz et al. (2019) is based on operations with large prime numbers, then the size of the transmitted data increases, which, in general, also increases the data transmission delay.

More recently in Lima et al. (2020), the authors introduce the idea of event-based encryption functions to ensure confidentiality of DESs, *i.e.*, only the sender and the intended receiver must be able to understand the transmitted data in the network. Language-based and transition-based encryption functions are defined and a test to verify the confidentiality of the DES is proposed for transition-based encryption functions using automaton formalism. However, in Lima et al. (2020), the asynchronous event-based framework is used without establishing a connection with the signals that are communicated to register the event occurrences. It is important to remark that, in automation systems, the transmitted data are bits, which are, in general, associated with sensor readings and commands to actuators. Thus, in order to implement an event-based cryptography it is necessary to introduce a codification of the system events and to bridge the event-based framework and the signal-based framework provided by the inputs and outputs of programmable logic controllers (PLC).

In this paper, we propose an encryption scheme that does not alter the transmission data size or structure in an automation network, which transmits only vectors of binary signals. In order to do so, we define events as changes in the binary signals transmitted in the communication channel, and present a cryptographic scheme, called event-based cryptography, that modifies the events in the application layer of the network so that the transmitted data structure remains the same. It is important to remark that in order to not change the frequency of the transmitted data it is necessary to use stream ciphers, *i.e.*, ciphers that process the input elements continuously, producing outputs, one element at a time. We adapt the ChaCha20 cipher presented in Bernstein (2008a), which is a variant of the awarded Salsa20/20 cipher presented in Bernstein (2008b). It is important to remark that the ChaCha20 cipher is recently attracting attention due to its deployment in several applications by Google (Mahdi et al., 2021). We present the implementation of the event-based cryptography using ChaCha20 cipher in an emulated automation network system. We also use the example to compare the size of the transmitted data and the time it takes to cipher, transmit, and decipher using ChaCha20 in comparison with the RSA cipher, a public-key cipher widely used in Information Technology (IT), implemented in a similar setting.

## 2. PRELIMINARIES

### *2.1 Notation and definitions*

Let $\mathbb{N}$ be the set of natural numbers and let $\mathbb{Z}_1 = \{0, 1\}$. The exclusive disjunction (XOR) operator, denoted by $\oplus$, is defined for two binary numbers $a, b \in \mathbb{Z}_1$ as $a \oplus b = 0$, if $a = b$, and $a \oplus b = 1$, if $a \neq b$. The XOR operation can be applied to two vectors of binary numbers $\underline{x}_1$ and $\underline{x}_2$ if the number of entries of $\underline{x}_1$ and $\underline{x}_2$ is equal. In this case, vector $\underline{x} = \underline{x}_1 \oplus \underline{x}_2$ is defined such that $x^i = x_1^i \oplus x_2^i$, where $x_j^i$ is the *i-th* element of $x_j$, for $j = 1, 2$. The number of elements of a vector $\underline{x}$ is denoted by $|\underline{x}|$.

### *2.2 Stream ciphers*

For any method of communication that may be accessible by non authorized entities, some level of encryption must be used to maintain data confidentiality (Stallings, 2006). Thus, the objective of cryptography is to transform a plain message $m$ into a cipher message $c$ such that only the receiver is able to recover the plain message $m$. In order to do so, it can be used asymmetric or symmetric ciphers, where in the former different keys are used for the sender and receiver to cipher and decipher the message, and, in the latter, the same key is used for encryption and decryption.

Some modern ciphers that are considered to be secure in Information Technology (IT) are based on the factorization of a number into prime numbers. The main drawback of using this technique is that, for ensuring data security, the prime numbers must be very large. As a consequence, the size of the transmitted message also increases. Thus, for transmitting one event in a communication channel, the cipher would increase the size of the transmitted data, causing large delays in the communication, which makes it not suitable for application in the automation network of cyber-physical systems.

The ciphers can also be partitioned into two groups: stream ciphers and block ciphers. Block ciphers process blocks of elements at a time, *i.e.*, if we consider binary vectors transmitted in a channel, a block cipher would gather several vectors and, then, encrypts these vectors as a block. After that, the ciphered block is transmitted to the receiver that decrypts the transmitted information. The problem with this approach is that, in general, in automation systems, it is necessary to observe the system events as soon as possible in order to do not add delays to the system, which is undesirable to the system control. Stream ciphers, on the other hand, process one element at a time, and therefore, are more appropriate to be used in automation networks. Thus, in this work, we propose the use of a symmetric stream cipher for encryption and decryption of the transmitted data in an automation network.

It is important to remark that there are several symmetric stream ciphers proposed in the literature that do not change the size of the ciphered data with respect to the plain data, such as ChaCha20. The ChaCha20 cipher is a variation of Salsa20/20, which has been considered one of the most secure and efficient ciphers in the eSTREAM project (Robshaw and Billet, 2008). This project was created with the objective to promote the design of stream ciphers with a particular emphasis on algorithms that would be either very fast in software or very resource-efficient in hardware.

### *2.3 ChaCha20*

The ChaCha20 cipher works by expanding a relatively small secret key $k_s$, known by the sender and receiver, into a keystream $K_S$. Keystream is the term used to refer to a large binary number that can be divided into several smaller binary numbers $k$, which are used in the encryption and decryption process. These smaller values are called encryption keys. Particularly, the ChaCha20 cipher uses

a secret key $k_s$ with 256 bits and 96 additional bits, called nonce, denoted as $\eta$, where $\eta$ cannot be repeated after generating a keystream $K_S$, *i.e.*, the nonce must be changed all the time a new keystream is generated. Moreover, in general, $\eta$ is a random or pseudo-random number, differently from the secret key $k_s$ which is a chosen number.

The ChaCha20 cipher is capable of generating, from the secret key $k_s$ and the nonce $\eta$, a total of $2^{32}$ different $4 \times 4$ matrices $M_q$, $q = 0, \ldots, 2^{32} - 1$, such that each entry of $M_q$ is a binary number with 32 bits that are used to generate the keystream $K_S$. In summary, both the sender and receiver, using the same 256-bits secret key and 96-bits nonce, are capable of generating the same keystream composed of $2^{41}$ bits, from which several encryption keys $k$ can be obtained as pieces of $K_S$. Then, each $k$ can be used for the encryption and decryption process of a single message. It is important to remark, that the number of bits of an encryption key $k$ is equal to the number of bits of the plain message $m$. For example, let us consider a system that transmits a message with 16 bits. Then, from the $2^{41}$ bits of the keystream $K_S$, 16 different bits of $K_S$ are chosen as $k$ each time a message is transmitted. Therefore, the sender and receiver are able to securely transmit $2^{37}$ messages, without changing the size or structure of the transmitted data. Notice that, in this example, if the system sends a message every 25 milliseconds, the communication channel would be able to operate securely for more than 100 years with the same secret key and nonce.

Notice that since $K_S$ has a large number of bits, in practice, only part of $K_S$ is generated initially, by computing matrices $M_q$ according to the size of the part of $K_S$ that must be stored. In this case, new matrices $M_q$ are computed when the part of $K_S$ that has already been computed is used in the encryption scheme. This avoids the storage of the complete $K_S$, which would require a large memory space to store it. An algorithm to compute the matrices $M_q$ is presented in Appendix A.

The cipher message $c$ is obtained using the ChaCha20 cipher by performing a XOR operation between the plain message $m$ and the current encryption key $k$, *i.e.*, $c = m \oplus k$. Since the sender and receiver can generate the same keystream $K_S$, then the receiver is capable of recovering the plain message by using the XOR operation between $c$ and $k$, *i.e.*, $m = c \oplus k$. The details of the ChaCha20 cipher are presented in Appendix A.

## 3. EVENT-BASED CRYPTOGRAPHIC SCHEME

In this paper, we address the problem of ensuring confidentiality in the automation network of a cyber-physical system. We consider that the communication is carried out using a wired or wireless network. The sender and receiver can represent different entities, *e.g.*, the sender may be an industrial plant and the receiver a controller or supervisor, and vice-versa. Thus, the $j$-th data transmitted from the sender to receiver is a vector of binary signal values $\underline{u}_j = [\alpha_1(j) \;\; \alpha_2(j) \;\; \ldots \;\; \alpha_n(j)]^T \in \mathbb{Z}_1^n$, which, in the case of automation systems, are, in general, formed of binary signals of sensors and actuators. We call $\underline{u}_j$ the plain vector.

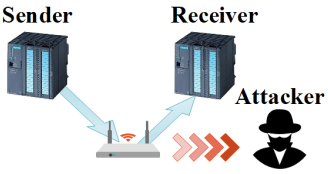


Fig. 1. Eavesdropping attack in the communication channel between sender and receiver.

We consider that the communication channel between sender and receiver is vulnerable to attacks, as shown in Figure 1, where the attacker can observe the data transmitted in this channel. The attacker eavesdrops the binary vector $\underline{u}_j$ with the objective of estimating the system state or its dynamic behavior. In this paper, we do not consider that the attacker can modify the data transmitted in the attacked channel, *i.e.*, the attacker performs only passive attacks (Stallings, 2006).

In order to ensure confidentiality of the transmitted data, several methods of encryption are proposed in IT, where the main objective is to keep the information secret, regardless of the size of the transmitted data. However, in automation networks, it is important to avoid the increase in the data size to do not delay the transmission of the information to the receiver. Thus, in this work, we propose an event-based cryptographic method that keeps the structure of the transmitted data, avoiding the increase in the transmission delay, *i.e.*, instead of transmitting the plain vector $\underline{u}_j \in \mathbb{Z}_1^n$, we will transmit a cipher vector $\underline{v}_j \in \mathbb{Z}_1^n$. In order to do so, we first define an event as any change in at least one of the entries of the plain vector. Let $\underline{u}_j$ and $\underline{u}_{j-1}$ be the $j$-th and $(j-1)$-th observed plain vectors, respectively, and consider that $\underline{u}_j \neq \underline{u}_{j-1}$. Then, an event $\sigma$ is coded by vector

$$\underline{\sigma} = \underline{u}_j \oplus \underline{u}_{j-1}.$$

Notice that the $i$-th entry of the event vector $\underline{\sigma}$, $\sigma_i$, is equal to one if there is a change from $\alpha_i(j-1)$ to $\alpha_i(j)$, and it is zero otherwise, *i.e.*, $\underline{\sigma}$ represents the changes in the values of the entries of the observed plain vector. Thus, the number of different events depends on the number of different observed plain vectors $\underline{u}_j$. It is important to remark that when $\underline{u}_j = \underline{u}_{j-1}$, no event is generated. In this case, we consider that the empty sequence $\varepsilon$ has been generated, which is coded as $\underline{u}_j \oplus \underline{u}_{j-1} = \underline{0}$.

*Remark 1.* A similar representation of events has been proposed in Moreira and Lesage (2019a,b) and de Souza et al. (2020) for the identification of DESs with the aim of fault detection. In the event representation proposed in these works, the authors consider that the change in the reading of an entry of vector $\underline{u}_j$ can be negative, when its value goes from 1 to 0, or positive, when its value goes from 0 to 1, distinguishing the falling and rising edges of the signal, respectively. In this paper, events are defined as changes in the entries of vector $\underline{u}_j$, independently if it is a rising or a falling edge of the binary signal, which allows the occurrence of two equal consecutive events. This facilitates the definition of the encryption and decryption methods proposed in this work. □

*Example 1.* Let us consider that the data transmitted in the communication channel is composed of three binary signals, *i.e.*, $\underline{u}_j = [\,\alpha_1(j) \;\; \alpha_2(j) \;\; \alpha_3(j)\,]^T$, and that

the following four plain vectors have been observed: $\underline{u}_0 = [0\ 0\ 0]^T$; $\underline{u}_1 = [1\ 0\ 0]^T$; $\underline{u}_2 = [1\ 0\ 0]^T$; and $\underline{u}_3 = [0\ 1\ 0]^T$. Then, when the plain vector changes from vector $\underline{u}_0$ to $\underline{u}_1$, an event $\sigma_1$, associated with the change of the first entry of the observed vector, is generated. This event is coded as $\underline{\sigma}_1 = \underline{u}_1 \oplus \underline{u}_0 = [1\ 0\ 0]^T$. In the sequel, no event is generated since $\underline{u}_2 = \underline{u}_1$, which implies that $\varepsilon$, coded as $\underline{u}_2 \oplus \underline{u}_1 = \underline{0}$, has been generated. Then, when we observe the change from vector $\underline{u}_2$ to $\underline{u}_3$, a new event $\sigma_2$ is generated. This event is coded as the event vector $\underline{\sigma}_2 = \underline{u}_3 \oplus \underline{u}_2 = [1\ 1\ 0]^T$. It is important to remark that since each event represents a change in at least one binary value of $\underline{u}_j$, without distinguishing a rising edge from a falling edge, then, if the value of $\underline{u}_3$ is modified such that $\underline{u}_3 = \underline{u}_0 = [0\ 0\ 0]^T$ and the sequence of observed vectors is $\underline{u}_0\underline{u}_1\underline{u}_2\underline{u}_3$, the associated sequence of events is $\sigma_1\varepsilon\sigma_1 = \sigma_1\sigma_1$. □

Let us consider the cryptographic scheme depicted in Figure 2, where $\underline{u}_j = [\,\alpha_1(j)\ \alpha_2(j)\ \ldots\ \alpha_n(j)\,]^T$ is the plain vector, $\underline{v}_j = [\,\gamma_1(j)\ \gamma_2(j)\ \ldots\ \gamma_n(j)\,]^T$ is the transmitted cipher vector, and $\underline{\sigma}$ is the event vector obtained when $\underline{u}_j$ and $\underline{u}_{j-1}$ are different. When an event $\underline{\sigma}$ is generated, it is ciphered as $\underline{\sigma}_c = \underline{\sigma} \oplus \underline{k}$, where $\underline{k} \in \mathbb{Z}_1^n$ is the vector formed with the elements of encryption key $k$. After computing $\underline{\sigma}_c$, the transmitted vector is modified to $\underline{v}_j = \underline{v}_{j-1} \oplus \underline{\sigma}_c$. Notice that when $\underline{u}_j$ and $\underline{u}_{j-1}$ are equal, then the empty sequence $\varepsilon$, coded as $\underline{0}$, is generated. In this case, the same cipher vector is transmitted, *i.e.*, $\underline{v}_j = \underline{v}_{j-1}$. Since $\underline{u}_0$ is the first observed plain vector, then we transmit the cipher vector $\underline{v}_0 = \underline{u}_0 \oplus \underline{k}$.

Since the size of each event is $|\underline{u}|$ bits, the event-based cipher needs to use $|\underline{u}|$ bits from $K_S$ to form a new vector $\underline{k}$ each time an event is observed. Thus, the encryption and decryption processes start using the first $|\underline{u}|$ bits from $K_S$ for the encryption and decryption of the first plain vector. Then, these bits are discarded in order to cipher and decipher the next event with the next $|\underline{u}|$ bits of $K_S$. If the entire keystream $K_S$ is used in the encryption process, then a new keystream with $2^{41}$ bits can be generated with a new secret $k_s$ and nonce $\eta$ as shown in Appendix A.

Notice that, the transmitted vector $\underline{v}_j$ can be observed by the attacker through an eavesdrop attack. However, without knowing $\underline{k}$, the attacker is not able to recover $\underline{u}_j$. On the other hand, the receiver knows $\underline{k}$ and is able to recover event $\underline{\sigma}$. After observing a variation between the transmitted vectors $\underline{v}_j$ and $\underline{v}_{j-1}$, the ciphered event $\underline{\sigma}_c = \underline{v}_j \oplus \underline{v}_{j-1}$ is computed, and then $\underline{\sigma}$ is obtained as $\underline{\sigma} = \underline{\sigma}_c \oplus \underline{k}$. Thus, the receiver, knowing the previous state $\underline{u}_{j-1}$, is able to recover the current state of the sender by making the XOR operation $\underline{u}_j = \underline{\sigma} \oplus \underline{u}_{j-1}$. It is important to remark that when $\underline{v}_j = \underline{v}_{j-1}$, then $\underline{u}_j = \underline{u}_{j-1}$.

Since the keystream $K_S$ is a pseudo-random large binary number, then it is possible that the observed event $\underline{\sigma}$ be equal to $\underline{k}$. In this case $\underline{\sigma}_c$ would be $\underline{0}$, which implies that $\underline{v}_j = \underline{v}_{j-1}$, which is undesirable since it leads to the same behavior as when $\varepsilon$ is generated. In order to circumvent this problem, we choose an event $\underline{\sigma}_\mu$ that never occurs in the system, and replace event $\underline{\sigma}$ with $\underline{\sigma}_\mu$ for the computation of the new cipher event $\underline{\sigma}_c = \underline{\sigma}_\mu \oplus \underline{k}$. In this case, the receiver recovers event $\underline{\sigma}_\mu$ by doing

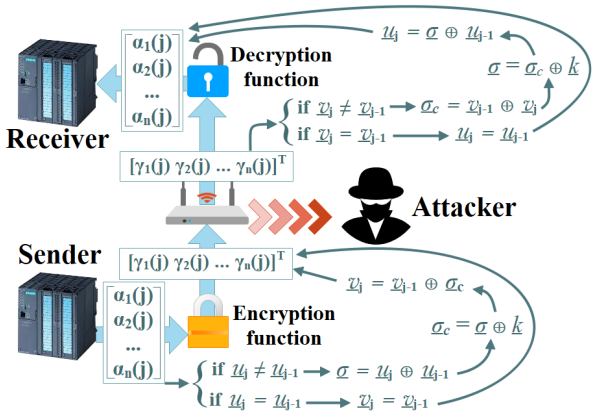


Fig. 2. Cryptographic scheme.

$\underline{\sigma} = \underline{\sigma}_c \oplus \underline{k} = \underline{\sigma}_\mu$, and detects that the correct event occurrence is $\underline{\sigma} = \underline{k}$, and use it to obtain $\underline{u}_j$.

*Remark 2.* Notice that it is almost impossible that all events in $\mathbb{Z}_1^n$ be generated in a large automation system. Thus, it is not difficult to define a vector $\underline{\sigma}_\mu$ to be used in the cryptographic scheme. □

The operation of the encryption procedure is presented in Algorithm 1. In Line 1, the encryption key $\underline{k}$ is computed from the keystream $K_S$. Then, in Line 2, the plain vector $\underline{u}_j$ is read. If it is the first observed plain vector $\underline{u}_0$, then $\underline{u}_0$ is encrypted as $\underline{v}_0 = \underline{u}_0 \oplus \underline{k}$ and transmitted. Otherwise, if a new plain vector $\underline{u}_j$ is observed, and it is different from the previous observed plain vector $\underline{u}_{j-1}$, $\underline{\sigma}$ is computed in Line 7 and ciphered, using key $\underline{k}$, in Line 8. If the cipher event vector $\underline{\sigma}_c$ is equal to $\underline{0}$, then the event that never occurs $\underline{\sigma}_\mu$ is ciphered in Line 10 to replace $\underline{\sigma}_c = \underline{0}$ with $\underline{\sigma}_c = \underline{\sigma}_\mu \oplus \underline{k}$. Then, in Line 11 the cipher vector $\underline{v}_j$ is computed and transmitted to the receiver. It is important to remark that, in Line 13, if $\underline{u}_j = \underline{u}_{j-1}$, then there is no need to actually calculate a cipher event since, by definition, it is equal to $\varepsilon$. Moreover, in this case, there is no need to update $\underline{k}$, since the key is not actually used, and also, the value of $\underline{v}_j$ is updated to $\underline{v}_j = \underline{v}_{j-1}$ in Line 14. Thus, the encryption process can skip steps and only wait for the next reading of the plain vector $\underline{u}_j$ in Line 2.

The decryption procedure is presented in Algorithm 2. In Line 1, a new encryption key $\underline{k}$ is computed. Then, in Line 2, a new observation of $\underline{v}_j$ is awaited. If it is the first observed cipher vector, then $\underline{u}_0$ is computed as $\underline{v}_0 \oplus \underline{k}$. Otherwise, if $\underline{v}_j$ is different from the previous observed vector $\underline{v}_{j-1}$, the cipher event $\underline{\sigma}_c$ is computed in Line 7, and $\underline{\sigma}$ is computed in Line 8. If $\underline{\sigma}$ is equal to $\underline{\sigma}_\mu$, then in Line 10, it is attributed $\underline{k}$ to $\underline{\sigma}$. Then, in Line 11, the plain vector $\underline{u}_j$ is computed, and the algorithm returns to Line 1. Notice that, in Line 13, if $\underline{v}_j = \underline{v}_{j-1}$, there is no need to decipher $\underline{v}_j$ since no modification has occurred, *i.e.*, $\underline{u}_j$ is equal to $\underline{u}_{j-1}$.

*Example 2.* In order to illustrate the encryption procedure presented in Algorithm 1, let us consider that we want to encrypt and transmit the three plain vectors of the system given by:

**Algorithm 1:** Encryption procedure

1 Obtain a new encryption key $\underline{k}$ from $K_S$
2 Read the plain vector $\underline{u}_j$
3 **if** $j = 0$ **then**
4 Transmit $\underline{v}_0 \leftarrow \underline{u}_0 \oplus \underline{k}$
5 Return to Line 1
6 **else if** $\underline{u}_j \neq \underline{u}_{j-1}$ **then**
7 $\underline{\sigma} \leftarrow \underline{u}_j \oplus \underline{u}_{j-1}$
8 $\underline{\sigma}_c \leftarrow \underline{\sigma} \oplus \underline{k}$
9 **if** $\underline{\sigma}_c = \underline{0}$ **then**
10 $\underline{\sigma}_c \leftarrow \underline{\sigma}_\mu \oplus \underline{k}$
11 $\underline{v}_j \leftarrow \underline{v}_{j-1} \oplus \underline{\sigma}_c$ and transmit $\underline{v}_j$
12 Return to Line 1
13 **else**
14 $\underline{v}_j \leftarrow \underline{v}_{j-1}$ and transmit $\underline{v}_j$
15 Return to Line 2

**Algorithm 2:** Decryption procedure

1 Obtain a new encryption key $\underline{k}$ from $K_S$
2 Read the cipher vector $\underline{v}_j$
3 **if** $j = 0$ **then**
4 $\underline{u}_0 \leftarrow \underline{v}_0 \oplus \underline{k}$
5 Return to Line 1
6 **else if** $\underline{v}_j \neq \underline{v}_{j-1}$ **then**
7 $\underline{\sigma}_c \leftarrow \underline{v}_j \oplus \underline{v}_{j-1}$
8 $\underline{\sigma} \leftarrow \underline{\sigma}_c \oplus \underline{k}$
9 **if** $\underline{\sigma} = \underline{\sigma}_\mu$ **then**
10 $\underline{\sigma} \leftarrow \underline{k}$
11 $\underline{u}_j \leftarrow \underline{u}_{j-1} \oplus \underline{\sigma}$
12 Return to Line 1
13 **else**
14 $\underline{u}_j \leftarrow \underline{u}_{j-1}$
15 Return to Line 2

$$\underline{u}_0 = [\,0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 0\,]^T,$$
$$\underline{u}_1 = [\,0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 0\,]^T,$$
$$\underline{u}_2 = [\,0\ 0\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 0\,]^T.$$

In Line 1 of the encryption procedure, it is necessary to obtain the first encryption key $\underline{k}$ from $K_S$, which requires the construction of part of the keystream $K_S$ using a 256-bit key and 96-bit nonce, which were omitted here due to lack of space. Since each matriz $M_q$, $q = 0, 1, \ldots, 2^{32} - 1$, provides 512 bits of $K_S$, then, in this example, for encrypting the message composed of the three vectors $\underline{u}_0$, $\underline{u}_1$ and $\underline{u}_2$, it is sufficient to construct only the first matrix $M_0$. After computing $M_0$, we can obtain, using the first 10 bits, the first encryption key $\underline{k}$ given by:

$$\underline{k} = [\,1\ 1\ 1\ 1\ 0\ 0\ 1\ 1\ 1\ 1\,]^T.$$

After observing the first plain vector $\underline{u}_0$, in Line 4 of Algorithm 1, the first cipher vector $\underline{v}_0$, computed as $\underline{v}_0 = \underline{u}_0 \oplus \underline{k} = [\,1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\,]^T$, is transmitted. Then, Algorithm 1 returns to Line 1, where a new key vector $\underline{k}$ is obtained from $M_0$ by using the next 10 bits given by:

$$\underline{k} = [\,1\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 0\,]^T.$$

After waiting for a new observation of a plain vector, *i.e.*, the sender reads the vector $\underline{u}_1$, it is checked whether this vector is equal to the previous value $\underline{u}_0$. Notice that, in this case $\underline{u}_0 = \underline{u}_1$, and therefore Algorithm 1 goes to Line 14, where the transmitted cipher vector is a copy of the previous cipher vector, *i.e.*, $\underline{v}_1 = \underline{v}_0$. It is important to remark that in this case Algorithm 1 goes to Line 2, and therefore, the value of the key vector $\underline{k}$ is not updated. Finally, when the sender observes the plain vector $\underline{u}_2$, event $\underline{\sigma} = \underline{u}_1 \oplus \underline{u}_2 = [\,0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\,]^T$ is obtained. In Line 8 of Algorithm 1 it is computed the cipher event $\underline{\sigma}_c = \underline{\sigma} \oplus \underline{k} = [\,1\ 0\ 0\ 1\ 0\ 1\ 1\ 0\ 0\ 0\,]^T$. Notice that, $\underline{\sigma}_c$ is different from $\underline{0}$, thus there is no need to use vector $\underline{\sigma}_\mu$. In Line 11 of Algorithm 1, it is calculated the new cipher vector $\underline{v}_2$, given by $\underline{v}_2 = \underline{\sigma}_c \oplus \underline{v}_1 = [\,0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 1\,]^T$. Finally, $\underline{v}_2$ is transmitted to the receiver and the encryption process returns to Line 1, where a new vector $\underline{k}$ is obtained and the system waits for a new reading of a plain vector.

In summary, the cipher vectors transmitted in the channel are:

$$\underline{v}_0 = [\,1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\,]^T,$$
$$\underline{v}_1 = [\,1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\,]^T,$$
$$\underline{v}_2 = [\,0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 1\,]^T.$$

It is important to remark that all cipher vectors $\underline{v}_0$, $\underline{v}_1$, and $\underline{v}_2$ are completely different from the plain vectors of the system, and therefore, an attacker would not be able to properly recover any information by observing the transmitted data.

On the other hand, the receiver can generate the same key vectors $\underline{k}$ as the sender, and therefore, the receiver can compute the plain vectors by observing the transmitted cipher vectors using Algorithm 2. In Line 1 of Algorithm 2, the first encryption key $\underline{k} = [\,1\ 1\ 1\ 1\ 0\ 0\ 1\ 1\ 1\ 1\,]^T$ is computed. Then, after observing the first cipher vector $\underline{v}_0$, in Line 4 of Algorithm 2, the receiver can compute $\underline{u}_0 = \underline{v}_0 \oplus \underline{k} = [\,0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 0\,]^T$. Then, $\underline{k}$ is updated in Line 1 to $\underline{k} = [\,1\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 0\,]^T$, and since $\underline{v}_1 = \underline{v}_0$, after observing $\underline{v}_1$, in Line 14, $\underline{u}_1 = \underline{u}_0$. Since $\underline{k}$ was not used, Algorithm 2 returns to Line 2 to wait for a new observation. After observing $\underline{v}_2 \neq \underline{v}_1$, in Line 7, the cipher event $\underline{\sigma}_c = \underline{v}_1 \oplus \underline{v}_2$ is computed, and then, in Line 8, $\underline{\sigma} = \underline{\sigma}_c \oplus \underline{k} = [\,0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\,]^T$ is recovered. Finally, the plain vector is computed in Line 11 as $\underline{u}_2 = \underline{u}_1 \oplus \underline{\sigma} = [\,0\ 0\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 0\,]^T$. □

In the next section, we present an analysis of the cryptographic scheme in an automation network. In order to do so, we emulate a Modbus TCP network in a computer and compared the ChaCha20 cipher with the RSA cipher, which is a public-key cipher widely used in IT.

## 4. ANALYSIS OF THE PROPOSED CRYPTOGRAPHIC SCHEME

### *4.1 Analysis Set up*

In order to analyze the proposed cryptographic scheme using the ChaCha20 cipher in an automation network, in this section we use plain vectors obtained from a simulation of an industrial sorting unit system that separates boxes according to their height, using the 3D simulation software Factory I/O, controlled by a Siemens S7-1200 programmable logic controller (PLC). The system is composed of six sensors and four actuators. Thus, we obtain a 10-bits vector $\underline{u}$ to represent the status of the system. The controlled system generated a total of 806,604 plain vectors $\underline{u}_j$, $j = 0; 1; \ldots; 806{,}603$. After obtaining the plain vectors, we have emulated a Modbus automation network with the objective of analyzing the encryption, decryption and transmission times of the proposed scheme, and to compare it with the RSA cipher. In order to do so, we emulated a Modbus communication in a computer running with Windows 10 Home 64 bits edition, with a processor Intel®Core(TM) i3-5th generation, 2.00 GHz, and 4GB of RAM, using Python version 3.9.7. The Modbus network was emulated using the Python library pyModbusTCP (Lefebvre, 2022; Roomi et al., 2020; Kang and Lee, 2019). In the Modbus set up, we have defined the server as the sender and the client as the receiver to transmit the cipher vectors in a Modbus channel.

After setting up the Modbus network, we have implemented the encryption scheme, proposed in Algorithm 1, in the server, and the decryption scheme, proposed in Algorithm 2, in the client, using Python, in the application layer of the automation network. Moreover, we set the reading and encryption of a new plain vector $\underline{u}_j$ by the server every 2.3 seconds, and the reading and decryption of the cipher vector by the client every 1 second.

### *4.2 Analysis*

In order to evaluate the viability of implementing the ChaCha20 cipher in an automation network, the 806,604 plain vectors were encrypted, transmitted, and deciphered using the emulated Modbus network. In this regard, the ChaCha20 cipher needs to compute keystream matrices $M_q$ to use, which can be carried out in the system idle time. It was computed two different times for obtaining part of the keystream. The first was the computation of a large part of the keystream $K_S$ containing 196,608 bits, obtained by using 384 matrices $M_q$. This part of $K_S$ was computed with average time 882.09 $ms$ in both sender and receiver. Notice that, this part of $K_S$ can be computed off-line and stored to be used in the encryption and decryption processes, 10 bits at a time, in a total of 19,660 encryptions and decryptions. Since the storage of 196,608 bits may not be viable for devices with low memory, then we also computed the time required for the computation of a single matrix $M_q$ with 512 bits which can be used for encrypting the first plain vector and the next 50 events. This computation were carried out with average time equal to 4.977 $ms$.

Considering that the keystream has already been obtained, as shown in Table 1, the average time for encrypting a message using the ChaCha20 cipher was 2.547 $ms$. The average time of transmitting the cipher vector from sender to receiver was 1.031 $ms$, and the average time to decipher the cipher vector in the receiver was 2.271 $ms$. Thus, the total average time used for encryption, transmission and decryption of a vector in the proposed cryptographic scheme was 5.849 $ms$.

Table 1. Average time of the cryptographic schemes using ChaCha20 and RSA ciphers.

| | RSA | ChaCha20 |
|---|---|---|
| Encryption time | 7.966 ms | 2.547 ms |
| Transmission time | 4.07 ms | 1.031 ms |
| Decryption time | 56.6 ms | 2.271 ms |

In order to compare the results with another encryption, we also implemented a RSA cipher in the same setting to transmit the system status. The plain vector $\underline{u}_j$ was directly encrypted using the RSA cipher in the sender (server), and deciphered in the receiver (client). In order to ensure security, the RSA cipher requires that the 10-bits plain vector be transformed into a 2000-bits cipher vector, which was performed with average time, according to Table 1, 7.966 $ms$, *i.e.*, 212% greater than the encryption time of the method using ChaCha20. The average transmission time of the RSA cipher was 4.07 $ms$, which is 294 % greater than the transmission time of the ChaCha20 cipher. Finally, the average decryption time using the RSA cipher was 56.6 $ms$, which is 2,392% greater than using the ChaCha20 cipher. These results shows that the cryptographic scheme using the ChaCha20 cipher is much more efficient than using the RSA cipher.

## 5. CONCLUSIONS

In this paper, we propose an event-based cryptographic scheme for ensuring data confidentiality, preventing a malicious agent from recovering the plain information transmitted between sender and receiver in an automation network of a cyber-physical system. In order to do so, we propose the use of the stream cipher ChaCha20, which is known to have high resistance to cryptanalysis. The encryption and decryption schemes do not alter the transmission data size or structure, which leads to an efficient cryptographic method to be used in automation networks, since it inserts a relatively small delay in the communication with respect to other cryptographic schemes. In order to show this fact, we present an example to illustrate the proposed event-based cryptographic scheme, and we compare it with a public-key cipher, called RSA, which is commonly used in Information Technology.

## REFERENCES


Alwan, A.A., Ciupala, M.A., Brimicombe, A.J., Ghorashi, S.A., Baravalle, A., and Falcarin, P. (2022). Data quality challenges in large-scale cyber-physical systems: A systematic review. *Information Systems*, 105, 101951.

Barcelos, R.J. and Basilio, J.C. (2021). Enforcing current-state opacity through shuffle and deletions of event observations. *Automatica*, 133, 109836.

Basilio, J.C., Hadjicostis, C.N., and Su, R. (2021). Analysis and control for resilience of discrete event systems:

Fault diagnosis, opacity and cyber security. *Foundations and Trends in Systems and Control*, 8(4), 285–443.

Bernstein, D.J. (2008a). ChaCha, a variant of Salsa20. In *Workshop Record of SASC*, volume 8, 3–5.

Bernstein, D.J. (2008b). The Salsa20 family of stream ciphers. In *New stream cipher designs*, 84–97. Springer.

de Souza, R.P., Moreira, M.V., and Lesage, J.J. (2020). Fault detection of discrete-event systems based on an identified timed model. *Control Engineering Practice*, 105, 104638.

Fritz, R., Fauser, M., and Zhang, P. (2019). Controller encryption for discrete event systems. In *2019 American Control Conference (ACC)*, 5633–5638. IEEE, Philadelphia, CA, USA.

Fritz, R. and Zhang, P. (2018). Modeling and detection of cyber attacks on discrete event systems. *IFAC-PapersOnLine*, 51(7), 285–290.

Gentry, C. (2009). *A fully homomorphic encryption scheme*. Stanford university.

Jacob, R., Lesage, J.J., and Faure, J.M. (2016). Overview of discrete event systems opacity: Models, validation, and quantification. *Annual Reviews in Control*, 41, 135–146.

Ji, Y., Wu, Y.C., and Lafortune, S. (2018). Enforcement of opacity by public and private insertion functions. *Automatica*, 93, 369–378.

Kang, S. and Lee, I. (2019). Implementation of PV monitoring system using Python. In *2019 21st International Conference on Advanced Communication Technology (ICACT)*, 453–455. IEEE.

Kurose, J.F. and Ross, K.W. (2011). *Computer networking: a top-down approach*. Addison Wesley.

Lafortune, S., Lin, F., and Hadjicostis, C.N. (2018). On the history of diagnosability and opacity in discrete event systems. *Annual Reviews in Control*, 45, 257–266.

Lefebvre, L. (2022). pyModbusTCP Documentation. `https://pymodbustcp.readthedocs.io/en/stable/`. [Online; accessed 22-February-2022].

Li, X., Hadjicostis, C.N., and Li, Z. (2021). Extended insertion functions for opacity enforcement in discrete event systems. *IEEE Transactions on Automatic Control*. doi: 10.1109/TAC.2021.3121249.

Lima, P.M., Alves, M.V.S., Carvalho, L.K., and Moreira, M.V. (2021). Security of cyber-physical systems: Design of a security supervisor to thwart attacks. *IEEE Transactions on Automation Science and Engineering*. doi:10.1109/TASE.2021.3076697.

Lima, P.M., Alves, M.V.S., Carvalho, L.K., and Moreira, M.V. (2019). Security Against Communication Network Attacks of Cyber-Physical Systems. *Journal of Control, Automation and Electrical Systems*, 30(1), 125–135.

Lima, P.M., Carvalho, L.K., and Moreira, M.V. (2020). Confidentiality of cyber-physical systems using event-based cryptography. In *21st IFAC World Congress 2020*, 1761–1766. Berlin, Germany.

Lin, F. (2011). Opacity of discrete event systems and its applications. *Automatica*, 47(3), 496–503.

Mahdi, M.S., Hassan, N.F., and Abdul-Majeed, G.H. (2021). An improved ChaCha algorithm for securing data on IoT devices. *SN Applied Sciences*, 3(4), 1–9.

Moreira, M.V. and Lesage, J.J. (2019a). Fault diagnosis based on identified discrete-event models. *Control Engineering Practice*, 91, 104101.

Moreira, M. and Lesage, J.J. (2019b). Discrete event system identification with the aim of fault detection. *Discrete Event Dynamic Systems*, 29, 191–209.

Robshaw, M. and Billet, O. (2008). *New stream cipher designs: the eSTREAM finalists*. Springer.

Roomi, M.M., Biswas, P.P., Mashima, D., Fan, Y., and Chang, E.C. (2020). False data injection cyber range of modernized substation system. In *2020 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm)*, 1–7. IEEE.

Song, H., Rawat, D.B., Jeschke, S., and Brecher, C. (2016). *Cyber-physical systems: foundations, principles and applications*. Morgan Kaufmann.

Stallings, W. (2006). *Cryptography and network security, 4/E*. Pearson Education India.

Tao, F., Qi, Q., Wang, L., and Nee, A. (2019). Digital Twins and Cyber–Physical Systems toward Smart Manufacturing and Industry 4.0: Correlation and Comparison. *Engineering*, 5(4), 653–661.

Tong, Y., Cai, K., and Giua, A. (2018). Decentralized Opacity Enforcement in Discrete Event Systems Using Supervisory Control. In *2018 57th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE)*, 1053–1058.

Yaacoub, J.P.A., Salman, O., Noura, H.N., Kaaniche, N., Chehab, A., and Malli, M. (2020). Cyber-physical systems security: Limitations, issues and future trends. *Microprocessors and microsystems*, 77, 103201.

Yin, X. and Lafortune, S. (2015). A new approach for synthesizing opacity-enforcing supervisors for partially-observed discrete-event systems. In *2015 American Control Conference (ACC)*, 377–383.

## Appendix A. CHACHA20 CIPHER

### *A.1 Notation and definitions*

Given two positive numbers $a$ and $n$, we have that $a$ modulo $n$, denoted as $a\ mod(n)$, is the remainder of the division of $a$ by $n$. In this paper, we use the operator $\boxplus$, to denote the addition modulo $2^{32}$, *i.e.*, $a \boxplus b = (a + b)\ mod(2^{32})$. We also denote the left-rotation operation for a 32-bits number as $\lll$, *i.e*, the operation of shifting all bits of a 32-bits number, one bit to the left, and placing the leftmost bit (most significant) as the rightmost bit of the 32-bits number. We also define the left-rotation operation for a given number $\ell \in \mathbb{N}$, such that given a 32-bits number $a$ then $a \lll \ell$ represents repeating the left-rotation operation of $a$, $\ell$ times.

### *A.2 Keystream generation*

We present in Algorithm 3 the function of a quarter-round operation that is used during the ChaCha20 keystream generation (Bernstein, 2008a).

**Algorithm 3:** $[a, b, c, d] =$**Quarter-round**$(a, b, c, d)$

**Input:** 32-bits numbers $a, b, c, d$
**Output:** modified 32-bits numbers $a, b, c, d$

1 $a \leftarrow a \boxplus b$
2 $d \leftarrow d \oplus a$
3 $d \leftarrow d \lll 16$
4 $c \leftarrow c \boxplus d$
5 $b \leftarrow b \oplus c$
6 $b \leftarrow b \lll 12$
7 $a \leftarrow a \boxplus b$
8 $d \leftarrow d \oplus a$
9 $d \leftarrow d \lll 8$
10 $c \leftarrow c \boxplus d$
11 $b \leftarrow b \oplus c$
12 $b \leftarrow b \lll 7$

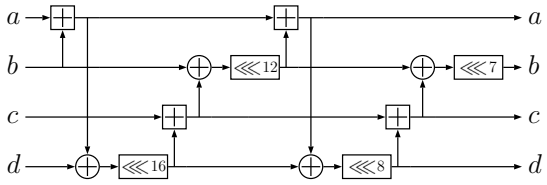


Fig. A.1. Quarter Round operation.

In order to illustrate the **Quarter-round** function, the operation over the inputs $a, b, c$ and $d$ are represented in Figure A.1. Notice that, the original values of $a, b, c$ and $d$ are updated by function **Quarter-round** by making left-rotation, XOR and addition modulo $2^{32}$ operations.

Matrices $M_q$, $q = 0, \ldots, 2^{32} - 1$, used for generating the keystream $K_S$, are computed according to Algorithm 4. In order to do so, in Line 1, $S$ is constructed as:

$$S = \begin{bmatrix} const_0 & const_1 & const_2 & const_3 \\ k_{s_0} & k_{s_1} & k_{s_2} & k_{s_3} \\ k_{s_4} & k_{s_5} & k_{s_6} & k_{s_7} \\ count & n_0 & n_1 & n_2 \end{bmatrix},$$

where $const_0$ , $const_1$, $const_2$, and $const_3$ are the binary numbers with 32 bits presented in Bernstein (2008a). The values of $k_{s_0}$, $k_{s_1}$, $k_{s_2}$, $k_{s_3}$, $k_{s_4}$, $k_{s_5}$, $k_{s_6}$, and $k_{s_7}$ are binary numbers with 32 bits obtained after partitioning the secret key $k_s$, that has 256 bits, into 8 parts, and *count* is a 32-bits counter. The values of $n_0$, $n_1$, and $n_2$ are binary numbers with 32 bits obtained after partitioning the nonce $\eta$, that has 96 bits, into three parts.

In Line 2 of Algorithm 4, we make a copy of matrix $S$, denoted by $S_0$. In order to facilitate the notation we will refer to each 32-bits long binary numbers of matrix $S_0$ as follows:

$$S_0 = \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix}.$$

After defining matrix $S_0$ we will modify it, generating a new matrix by following the operations from Lines 5 to 12. The operations:

$[s_{0,0}, s_{1,0}, s_{2,0}, s_{3,0}] =$ **Quarter-round**$(s_{0,0}, s_{1,0}, s_{2,0}, s_{3,0})$
$[s_{0,1}, s_{1,1}, s_{2,1}, s_{3,1}] =$ **Quarter-round**$(s_{0,1}, s_{1,1}, s_{2,1}, s_{3,1})$
$[s_{0,2}, s_{1,2}, s_{2,2}, s_{3,2}] =$ **Quarter-round**$(s_{0,2}, s_{1,2}, s_{2,2}, s_{3,2})$
$[s_{0,3}, s_{1,3}, s_{2,3}, s_{3,3}] =$ **Quarter-round**$(s_{0,3}, s_{1,3}, s_{2,3}, s_{3,3})$

constitute a column round, and the operations:

$[s_{0,0}, s_{1,1}, s_{2,2}, s_{3,3}] =$ **Quarter-round**$(s_{0,0}, s_{1,1}, s_{2,2}, s_{3,3})$
$[s_{0,1}, s_{1,2}, s_{2,3}, s_{3,0}] =$ **Quarter-round**$(s_{0,1}, s_{1,2}, s_{2,3}, s_{3,0})$
$[s_{0,2}, s_{1,3}, s_{2,0}, s_{3,1}] =$ **Quarter-round**$(s_{0,2}, s_{1,3}, s_{2,0}, s_{3,1})$
$[s_{0,3}, s_{1,0}, s_{2,1}, s_{3,2}] =$ **Quarter-round**$(s_{0,3}, s_{1,0}, s_{2,1}, s_{3,2})$

constitute a diagonal round.

**Algorithm 4:** Construction of Matrix $M_q$

**Input:** $k_s = k_{s_0} \ldots k_{s_7}$, $\eta = n_0 n_1 n_2$, *count*, $const_0$, $const_1$, $const_2$, $const_3$
**Output:** $M_q$, where $q$ is the decimal representation of *count*

1 $S = \begin{bmatrix} const_0 & const_1 & const_2 & const_3 \\ k_{s_0} & k_{s_1} & k_{s_2} & k_{s_3} \\ k_{s_4} & k_{s_5} & k_{s_6} & k_{s_7} \\ count & n_0 & n_1 & n_2 \end{bmatrix}$
2 $S_0 = \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} \leftarrow S$
3 $h = 1$
4 **while** $h \leq 10$ **do**
5 $\quad [s_{0,0}, s_{1,0}, s_{2,0}, s_{3,0}] =$ **Quarter-round**$(s_{0,0}, s_{1,0}, s_{2,0}, s_{3,0})$
6 $\quad [s_{0,1}, s_{1,1}, s_{2,1}, s_{3,1}] =$ **Quarter-round**$(s_{0,1}, s_{1,1}, s_{2,1}, s_{3,1})$
7 $\quad [s_{0,2}, s_{1,2}, s_{2,2}, s_{3,2}] =$ **Quarter-round**$(s_{0,2}, s_{1,2}, s_{2,2}, s_{3,2})$
8 $\quad [s_{0,3}, s_{1,3}, s_{2,3}, s_{3,3}] =$ **Quarter-round**$(s_{0,3}, s_{1,3}, s_{2,3}, s_{3,3})$
9 $\quad [s_{0,0}, s_{1,1}, s_{2,2}, s_{3,3}] =$ **Quarter-round**$(s_{0,0}, s_{1,1}, s_{2,2}, s_{3,3})$
10 $\quad [s_{0,1}, s_{1,2}, s_{2,3}, s_{3,0}] =$ **Quarter-round**$(s_{0,1}, s_{1,2}, s_{2,3}, s_{3,0})$
11 $\quad [s_{0,2}, s_{1,3}, s_{2,0}, s_{3,1}] =$ **Quarter-round**$(s_{0,2}, s_{1,3}, s_{2,0}, s_{3,1})$
12 $\quad [s_{0,3}, s_{1,0}, s_{2,1}, s_{3,2}] =$ **Quarter-round**$(s_{0,3}, s_{1,0}, s_{2,1}, s_{3,2})$
13 $\quad h \leftarrow h + 1$
14 $M_q = S \boxplus \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix}$

The process of performing a column round followed by a diagonal round is repeated 10 times in the loop of Line 4, *i.e.*, there are a total of 10 alternating column rounds and diagonal rounds to modify matrix $S_0$. After that, matrix $M_q$ is formed in Line 14 by performing the addition modulo $2^{32}$ of each 32-bits element of the original matrix $S$ with the corresponding element $s_{i,j}$, for $i, j = 0, 1, 2, 3$.

It is important to remark that Algorithm 4 produces a 512-bits matrix $M_q$ as an output. This process can be repeated with a different value for the counter *count* to generate a new 512-bits matrix $M_q$ that can be used with the same secret key $k_s$ and nonce $\eta$. Thus, the number of bits of the part of the keystream $K_S$ that we want to compute, provides the number of times Algorithm 4

must be executed generating different matrices $M_q$. After obtaining the matrix $M_q$, part of keystream $K_S$ is obtained by converting the rows of $M_q$ into a single binary number with 512 bits concatenating the rows side by side.