Uso da cifra de fluxo ChaCha20 em redes de automação

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

[*] *Departamento de Automação e Sistemas, Universidade Federal de Santa Catarina, Campus Trindade, Florianópolis, 88.040-900, SC, Brazil*
[**] *COPPE - Programa de Engenharia Elétrica, Universidade Federal do Rio de Janeiro, Cidade Universitária, Ilha do Fundão, Rio de Janeiro, 21.945-970, RJ, Brazil,*
[***] *COPPE - Programa de Engenharia de Sistemas e Computação, Universidade Federal do Rio de Janeiro, Cidade Universitária, Ilha do Fundão, Rio de Janeiro, 21.945-970, RJ, Brazil,*
Emails: publio.lima@ufsc.br, carloskpereira.07@poli.ufrj.br, cmicelifarias@cos.ufrj.br, lilian.carvalho@poli.ufrj.br, moreira.mv@poli.ufrj.br

Abstract: With the advance of the Industry 4.0, it is critical to ensure security against cyber attacks even in the field level of the industry. In this paper, we consider cyber attacks in automation networks, where an attacker eavesdrops communication channels in order to gather information about the system behavior. We propose a cryptographic scheme used in the application layer of the network which encrypt events without altering the size or structure of the transmitted data. This scheme is denoted by event-based cryptography, where an event is defined as any change in the binary vector transmitted in the network. In order to do so, we propose the use of a cryptography scheme called ChaCha20, which is known to have a high resistance to cryptanalysis.

Resumo: Com o avanço da Indústria 4.0, torna-se fundamental garantir a segurança contra ataques cibernéticos mesmo nos níveis de dispositivos de campo da planta. Neste trabalho, considera-se ataques cibernéticos em redes de automação em que o atacante espiona os canais de comunicação com o objetivo de conseguir informações sobre o comportamento do sistema. Assim, é proposto um sistema de criptografia na camada de aplicação de redes que encripta eventos sem alterar o tamanho ou a estrutura dos dados transmitidos. Esse esquema é chamado de criptografia baseada em eventos, em que um evento é definido como uma mudança no vetor binário transmitido na rede. Para tanto, é usada uma cifra denominada ChaCha20, que tem como característica principal a alta resistência à criptoanálise.

Keywords: Cyber Security; Cryptography; Discrete-Event Systems; Automation networks; Stream ciphers.

Palavras-chaves: Segurança cibernética; Criptografia; Sistemas a eventos discretos; Redes de automação; Cifras de Fluxo.

1. INTRODUÇÃO

Com o avanço da Indústria 4.0 torna-se necessário encontrar soluções para garantir a segurança contra ataques cibernéticos em sistemas industriais, mesmo para redes de campo (Yaacoub et al., 2020; Lima et al., 2021; Alves et al., 2022). Neste trabalho, são considerados ataques cibernéticos em que o atacante espiona os canais de comunicação de uma rede de automação com o objetivo de conseguir informações sobre o comportamento do sistema.

No contexto de Sistemas a Eventos Discretos (SED), para a segurança contra ataques cujo objetivo é descobrir um comportamento secreto do sistema, normalmente são usadas técnicas de ofuscação (Lafortune et al., 2018; Ji et al., 2018; Barcelos and Basilio, 2021; Li et al., 2021; Basilio et al., 2021). Outro método de proteger dados em canais de comunicação é o uso da criptografia (Stallings, 2006; Kurose and Ross, 2011; Fritz et al., 2019; Lima et al., 2020). Em Fritz et al. (2019), os autores propõem o uso de um esquema de criptografia baseado em operações com números primos grandes. Esse método leva a um aumento

⋆ Esse trabalho foi parcialmente financiado pelo CNPq - processo número 305267/2018-3, 431307/2018-0, e 436672/2018-9, FAPERJ, e pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de financiamento 001.

do tamanho do dado transmitido, o que, em geral, também aumenta o atraso da transmissão de dados.

Em Lima et al. (2020), é introduzida a ideia de funções de criptografia baseadas em eventos para garantir a confidencialidade de SEDs, ou seja, que apenas o emissor e o receptor entendam os dados transmitidos em uma rede (Stallings, 2006). Por outro lado, em Lima et al. (2020), a estrutura baseada em eventos assíncronos é usada sem estabelecer uma conexão com os sinais transmitidos para registrar a ocorrência de eventos. É importante ressaltar que, em sistemas de automação, os dados transmitidos são bits que, em geral, são associados a leituras de sensores ou comandos para os atuadores. Assim, para implementar uma criptografia baseada em eventos, é necessário introduzir uma codificação dos eventos do sistema e vincular a estrutura baseada em eventos com os sinais provenientes das entradas e saídas do controlador lógico programável (CLP).

Neste trabalho é proposto um esquema de criptografia que não altera o tamanho ou a estrutura dos dados em uma rede de automação, que transmite apenas vetores de sinais binários. Para tanto, eventos são definidos como mudanças nos sinais binários transmitidos no canal de comunicação e apresenta-se um sistema de criptografia, chamado de criptografia baseada em eventos, que modifica os eventos na camada de aplicação da rede. É importante ressaltar que para não mudar a frequência dos dados transmitidos é necessário usar criptografias de fluxo, ou seja, criptografias que processam os elementos de entrada continuamente, um elemento por vez. Neste trabalho, o algoritmo de criptografia ChaCha20 apresentado em Bernstein (2008a), que é uma variação da premiada cifra Salsa20 apresentada em Bernstein (2008b) e Robshaw and Billet (2008), é adaptado. É importante ressaltar que a cifra ChaCha20 tem atraído atenção atualmente por possuir alta resistência à criptoanálise e por ser usada em diversas aplicações da empresa Google (Mahdi et al., 2021).

Este trabalho está organizado da seguinte forma. Na seção 2, notações e definições básicas são apresentadas, bem como são definidas as cifras de fluxo e a cifra ChaCha20. Na seção 3, o esquema de criptografia baseada em eventos é apresentado, e um exemplo é utilizado para ilustrar os algoritmos propostos. Finalmente, na seção 4, as conclusões são apresentadas.

2. FUNDAMENTAÇÃO TEÓRICOS

2.1 Notação e definições

Seja $\mathbb{N}$ o conjunto dos números naturais e seja $\mathbb{Z}_1 = \{0, 1\}$. O operador disjunção exclusiva (XOR), $\oplus$, é definido para dois números binários $a, b \in \mathbb{Z}_1$ como $a \oplus b = 0$, se $a = b$, e $a \oplus b = 1$, se $a \neq b$. O operador XOR pode ser aplicado em dois vetores de números binários $\underline{x}_1$ e $\underline{x}_2$ se o número de entradas de $\underline{x}_1$ e $\underline{x}_2$ forem iguais. Nesse caso, o vetor $\underline{x} = \underline{x}_1 \oplus \underline{x}_2$ é definido tal que $x^i = x_1^i \oplus x_2^i$, em que x_j^i denota o *i-ésimo* elemento de x_j, com $j = 1, 2$. O número de elementos do vetor $\underline{x}$ é denotado por $|\underline{x}|$.

2.2 Cifras de fluxo

Para qualquer método de comunicação que possa ser acessado por entidades não autorizadas, algum nível de encriptação deve ser usado para manter a confidencialidade dos dados (Stallings, 2006). O objetivo da criptografia é transformar uma mensagem clara m em uma mensagem cifrada c tal que somente o receptor é capaz de recuperar a mensagem clara m.

Neste trabalho, propõe-se o uso de uma cifra de fluxo simétrica para cifrar e decifrar os dados transmitidos em uma rede de automação. Cifras simétricas são aquelas em que a chave de encriptação e de decifração são iguais, e cifras de fluxo são cifras que processam um elemento por vez. É importante ressaltar que existem diversas cifras simétricas propostas na literatura que não alteram o tamanho do dado cifrado em relação à mensagem clara, como é o caso do algoritmo de criptografia ChaCha20 (Bernstein, 2008a). A cifra ChaCha20 é uma variação de outra cifra denominada Salsa20/20 (Bernstein, 2008b), que foi considerada uma das mais seguras e eficientes no projeto eSTREAM (Robshaw and Billet, 2008). Esse projeto foi criado com o objetivo de incentivar a criação de cifras de fluxo com foco em algoritmos que fossem muito rápidos em software ou muito eficazes em hardware.

2.3 ChaCha20

A cifra ChaCha20 funciona expandindo uma chave secreta relativamente pequena k_s, conhecida pelo emissor e pelo receptor, em um *keystream* K_S. *Keystream* é o termo usado para se referir a um grande número binário que pode ser dividido em vários números binários menores k, que são usados no processo de encriptação e decifração. Esses valores menores são chamados de chaves de criptografia. Particularmente, a cifra ChaCha20 usa uma chave secreta com 256 bits e 96 bits adicionais, chamado *nonce*, em que o *nonce* não pode ser repetido após a geração de um *keystream* K_S, ou seja, o *nonce* deve ser alterado toda vez que um novo *keystream* é gerado. Várias cifras usam números *nonce* para evitar ataques de repetição. Além disso, em geral, o *nonce* é um número aleatório ou pseudo-aleatório, diferentemente da chave secreta que é um número escolhido. Neste artigo, o *nonce* é denotado como η.

A cifra ChaCha20 é capaz de gerar, a partir da chave secreta k_s e do *nonce* η, um total de 2^{32} matrizes 4×4 diferentes M_q, $q = 0, \ldots, 2^{32} - 1$, de modo que cada elemento de M_q seja um número binário com 32 bits que são usados para gerar o *keystream* K_S. Em resumo, tanto o emissor quanto o receptor, usando a mesma chave secreta de 256 bits e *nonce* de 96 bits, são capazes de gerar o mesmo *keystream* composto de 2^{41} bits, dos quais várias chaves de criptografia k podem ser obtidos tomando-se pedaços de K_S. Então, cada k pode ser usado para o processo de criptografia e decifração de uma única mensagem. É importante observar que o número de bits de uma chave de criptografia k é igual ao número de bits da mensagem clara m. Por exemplo, considerando um sistema que transmite uma mensagem com 16 bits. Então, dos 2^{41} bits do *keystream* K_S, 16 bits diferentes de K_S podem ser escolhidos como k cada vez que uma mensagem

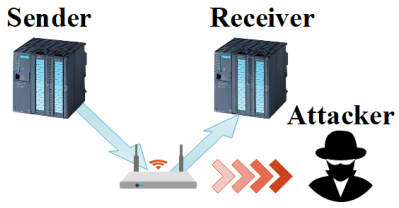

Figura 1. Ataque de espionagem no canal de comunicação entre emissor e receptor.

é transmitida. Portanto, o emissor e o receptor podem transmitir com segurança 2^{37} mensagens, sem alterar o tamanho ou a estrutura dos dados transmitidos. Observe que, neste exemplo, se o sistema enviar uma mensagem a cada 25 milissegundos, o canal de comunicação poderá operar com segurança por mais de 100 anos com a mesma chave secreta e *nonce*.

Observe que, como K_S possui um grande número de bits, na prática, apenas parte de K_S é gerada inicialmente, calculando as matrizes M_q de acordo com o tamanho da parte de K_S que deve ser armazenada. Neste caso, novas matrizes M_q são computadas quando a parte de K_S que já foi computada é utilizada no esquema de criptografia. Isso evita o armazenamento do K_S completo, o que exigiria um grande espaço de memória para armazená-lo. Um algoritmo para calcular as matrizes M_q é apresentado no Apêndice A.

A mensagem cifrada c é obtida usando a cifra ChaCha20 executando uma operação XOR entre a mensagem clara m e a chave de criptografia atual k, *i.e.*, $c = m \oplus k$. Como o emissor e o receptor podem gerar o mesmo *keystream* K_S, o receptor é capaz de recuperar a mensagem clara usando a operação XOR entre c e k, *i.e.*, $m = c \oplus k$. Os detalhes da cifra ChaCha20 são apresentados no Apêndice A.

3. ESQUEMA DE CRIPTOGRAFIA BASEADA EM EVENTOS

Neste artigo, aborda-se o problema de garantir a confidencialidade em uma rede de automação. Considera-se que a comunicação é realizada através de uma rede cabeada ou sem fio. O emissor e o receptor podem representar entidades diferentes, por exemplo, o emissor pode ser uma planta industrial e o receptor um controlador ou supervisor, e vice-versa. Assim, o *j-ésimo* dado transmitido do emissor para o receptor é um vetor de sinais binários $\underline{u}_j = [\alpha_1(j)\ \ \alpha_2(j)\ \ \ldots\ \ \alpha_n(j)]^T \in \mathbb{Z}_1^n$, que, no caso de sistemas de automação, são, em geral, formados por sinais binários de sensores e atuadores. O vetor $\underline{u}_j$ é denominado de vetor claro.

Considera-se que o canal de comunicação entre emissor e receptor é vulnerável a ataques, como mostrado na Figura 1, onde o invasor pode observar os dados transmitidos neste canal. O invasor intercepta o vetor binário $\underline{u}_j$ com o objetivo de estimar o estado do sistema ou seu comportamento dinâmico. Neste trabalho, não é considerado que o invasor possa modificar os dados transmitidos no canal atacado, ou seja, o invasor realiza apenas ataques passivos (Stallings, 2006).

Para garantir a confidencialidade dos dados transmitidos, diversos métodos de criptografia são propostos na área de Tecnologia da Informação (TI), onde o objetivo principal é manter o sigilo das informações, independentemente do tamanho dos dados transmitidos. Porém, em redes de automação, é importante evitar o aumento do tamanho dos dados para não atrasar a transmissão das informações ao receptor. Assim, neste trabalho, é proposto um método de criptografia baseado em eventos que mantém a estrutura dos dados transmitidos, evitando o aumento do atraso de transmissão, ou seja, ao invés de transmitir o vetor claro $\underline{u}_j \in \mathbb{Z}_1^n$, é transmitido um vetor cifrado $\underline{v}_j \in \mathbb{Z}_1^n$. Para isso, primeiro é definido um evento como qualquer mudança em pelo menos uma das entradas do vetor claro. Sejam $\underline{u}_{j-1}$ e $\underline{u}_j$ dois vetores claros observados consecutivamente, e considere que $\underline{u}_j \neq \underline{u}_{j-1}$. Então, um evento σ é codificado pelo vetor

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

Note que a i-ésima entrada do vetor de eventos $\underline{\sigma}$, σ_i, é igual a um se houver uma mudança de $\alpha_i(j-1)$ para $\alpha_i(j)$, e é zero caso contrário, ou seja, $\underline{\sigma}$ representa as mudanças nos valores das entradas do vetor claro observado. Assim, o número de eventos diferentes depende do número de vetores claros diferentes observados $\underline{u}_j$. É importante ressaltar que, quando $\underline{u}_j = \underline{u}_{j-1}$, nenhum evento é gerado. Neste caso, considera-se que foi gerada a sequência vazia ε, que é codificada como $\underline{u}_j \oplus \underline{u}_{j-1} = \underline{0}$.

Observação 1. Uma representação similar de eventos foi proposta em Moreira and Lesage (2019a,b) e de Souza et al. (2020) para a identificação de SEDs com o objetivo de detecção de falhas. Na representação de eventos proposta nesses trabalhos, os autores consideram que a mudança na leitura de uma entrada do vetor $\underline{u}_j$ pode ser negativa, quando seu valor passa de 1 a 0, ou positiva, quando seu valor passa de 0 a 1, distinguindo as bordas de descida e de subida do sinal, respectivamente. Neste trabalho, eventos são definidos como mudanças nas entradas do vetor $\underline{u}_j$, independente se for uma borda de subida ou de descida do sinal binário, o que permite a ocorrência de dois eventos iguais consecutivos. Isso facilita a definição dos métodos de cifragem e decifração propostos neste trabalho. □

Exemplo 1. Sejam os dados transmitidos no canal de comunicação compostos por três sinais binários, ou seja, $\underline{u}_j = [\,\alpha_1(j)\ \alpha_2(j)\ \alpha_3(j)\,]^T$, e que os quatro vetores claros a seguir foram observados: $\underline{u}_0 = [\,0\ 0\ 0\,]^T$; $\underline{u}_1 = [\,1\ 0\ 0\,]^T$; $\underline{u}_2 = [\,1\ 0\ 0\,]^T$; e $\underline{u}_3 = [\,0\ 1\ 0\,]^T$. Então, quando o vetor claro muda do vetor $\underline{u}_0$ para $\underline{u}_1$, um evento σ_1, associado à mudança da primeira entrada do vetor observado, é gerado. Esse evento é codificado como $\underline{\sigma}_1 = \underline{u}_1 \oplus \underline{u}_0 = [\,1\ 0\ 0\,]^T$. Em seguida, nenhum evento é gerado já que $\underline{u}_2 = \underline{u}_1$, o que implica que ε, codificado como $\underline{u}_2 \oplus \underline{u}_1 = \underline{0}$, foi gerado. Então, quando observa-se a mudança do vetor $\underline{u}_2$ para $\underline{u}_3$, um novo evento σ_2 é gerado. Esse evento é codificado como o vetor de eventos $\underline{\sigma}_2 = \underline{u}_3 \oplus \underline{u}_2 = [\,1\ 1\ 0\,]^T$. É importante observar que como cada evento representa uma mudança em pelo menos um valor binário de $\underline{u}_j$, sem distinguir uma borda de subida de uma borda de descida, então, se o valor de $\underline{u}_3$ for modificado de tal forma que $\underline{u}_3 = \underline{u}_0 = [0\ 0\ 0]^T$ e a sequência de vetores observados é $\underline{u}_0\underline{u}_1\underline{u}_2\underline{u}_3$, a sequência de eventos associada é $\sigma_1\varepsilon\sigma_1 = \sigma_1\sigma_1$. □

Considere o esquema de criptografia representado na Figura 2, em que $\underline{u}_j = [\,\alpha_1(j)\ \alpha_2(j)\ \ldots\ \alpha_n(j))\,]^T$ é o vetor

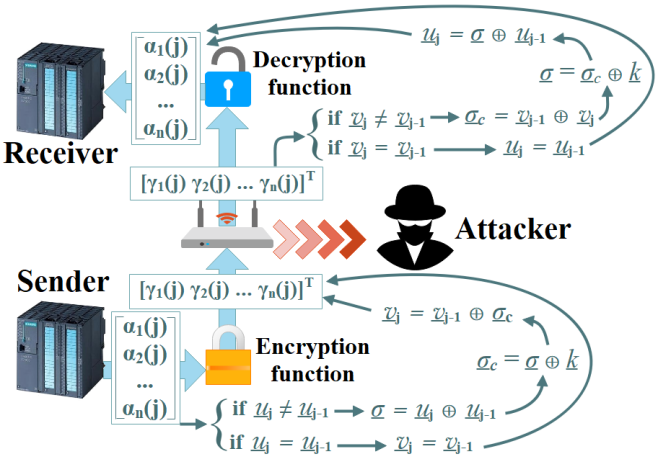

Figura 2. Esquema de criptografia.

claro, $\underline{v}_j = [\gamma_1(j)\ \gamma_2(j)\ \dots\ \gamma_n(j)]^T$ é o vetor cifrado transmitido e $\underline{\sigma}$ é o vetor de eventos obtido quando $\underline{u}_j$ e $\underline{u}_{j-1}$ são diferentes. Quando um evento $\underline{\sigma}$ é gerado, ele é criptografado como $\underline{\sigma}_c = \underline{\sigma} \oplus \underline{k}$, em que $\underline{k} \in \mathbb{Z}_1^n$ é o vetor formado com os elementos da chave de criptografia k. Após computar $\underline{\sigma}_c$, o vetor transmitido é modificado para $\underline{v}_j = \underline{v}_{j-1} \oplus \underline{\sigma}_c$. Observe que quando $\underline{u}_j$ e $\underline{u}_{j-1}$ são iguais, então a sequência vazia ε, codificada como $\underline{0}$, é gerada. Nesse caso, o mesmo vetor cifrado é transmitido, ou seja, $\underline{v}_j = \underline{v}_{j-1}$. Como $\underline{u}_0$ é o primeiro vetor claro observado, então o vetor cifrado $\underline{v}_0 = \underline{u}_0 \oplus \underline{k}$ é transmitido.

Como o tamanho de cada evento é $|\underline{u}|$ bits, a cifra baseada em eventos precisa usar $|\underline{u}|$ bits de K_S para formar um novo vetor $\underline{k}$ cada vez que um evento é observado. Assim, os processos de criptografia e decifração usam os primeiros $|\underline{u}|$ bits de K_S para a criptografia e decifração do primeiro vetor claro. Em seguida, esses bits são descartados para cifrar e decifrar o próximo evento com os próximos $|\underline{u}|$ bits de K_S. Se todo o *keystream* K_S for usado no processo de criptografia, então um novo *keystream* com 2^{41} bits pode ser gerado com uma nova chave secreta k_s e um novo *nonce* η como mostrado no Apêndice A.

Note que o vetor transmitido $\underline{v}_j$ pode ser observado pelo atacante através de um ataque de espionagem. No entanto, sem saber $\underline{k}$, o atacante não consegue recuperar $\underline{u}_j$. Por outro lado, o receptor conhece $\underline{k}$ e é capaz de recuperar o evento $\underline{\sigma}$. Após observar uma variação entre os vetores transmitidos $\underline{v}_j$ e $\underline{v}_{j-1}$, o evento cifrado $\underline{\sigma}_c = \underline{v}_j \oplus \underline{v}_{j-1}$ é calculado e então $\underline{\sigma}$ é obtido fazendo $\underline{\sigma} = \underline{\sigma}_c \oplus \underline{k}$. Assim, o receptor, conhecendo o estado anterior $\underline{u}_{j-1}$, é capaz de recuperar o estado atual do emissor fazendo a operação XOR $\underline{u}_j = \underline{\sigma} \oplus \underline{u}_{j-1}$. É importante ressaltar que quando $\underline{v}_j = \underline{v}_{j-1}$, então $\underline{u}_j = \underline{u}_{j-1}$.

Como o *keystream* K_S é um número binário grande pseudo-aleatório, então é possível que o evento observado $\underline{\sigma}$ seja igual a $\underline{k}$. Nesse caso $\underline{\sigma}_c$ seria $\underline{0}$, o que implica que $\underline{v}_j = \underline{v}_{j-1}$, o que é indesejável, pois leva ao mesmo comportamento de quando ε é gerado. Para contornar esse problema, é escolhido um evento $\underline{\sigma}_\mu$ nunca ocorrente no sistema para substituir o evento $\underline{\sigma}$ para o cálculo do novo evento cifrado $\underline{\sigma}_c = \underline{\sigma}_\mu \oplus \underline{k}$. Neste caso, o receptor recupera o evento $\underline{\sigma}_\mu$ fazendo $\underline{\sigma} = \underline{\sigma}_c \oplus \underline{k} = \underline{\sigma}_\mu$, e detecta a ocorrência do evento é $\underline{\sigma} = \underline{k}$, que é usado para obter $\underline{u}_j$.

Observação 2. Note que é quase impossível que todos os eventos em $\mathbb{Z}_1^n$ sejam gerados em um sistema de automação grande. Assim, não é difícil definir um vetor $\underline{\sigma}_\mu$ a ser utilizado no esquema de criptografia. □

O procedimento de criptografia é apresentado no Algoritmo 1. Na linha 1, a chave de criptografia $\underline{k}$ é calculada a partir da chave de fluxo K_S. Então, na linha 2, o vetor claro $\underline{u}_j$ é lido. Se for o primeiro vetor claro observado $\underline{u}_0$, então $\underline{u}_0$ é criptografado como $\underline{v}_0 = \underline{u}_0 \oplus \underline{k}$ e, em seguida, transmitido. Caso contrário, se um novo vetor claro $\underline{u}_j$ for observado e for diferente do vetor claro observado anteriormente $\underline{u}_{j-1}$, $\underline{\sigma}$ é calculado na linha 7 e cifrado, usando a chave $\underline{k}$, na linha 8. Se o vetor de evento cifrado $\underline{\sigma}_c$ for igual a $\underline{0}$, então o evento nunca ocorrente $\underline{\sigma}_\mu$ é cifrado na linha 10 para substituir $\underline{\sigma}_c = \underline{0}$ por $\underline{\sigma}_c = \underline{\sigma}_\mu \oplus \underline{k}$. Então, na linha 11, o vetor cifrado $\underline{v}_j$ é calculado e transmitido ao receptor. É importante ressaltar que, na linha 13, se $\underline{u}_j = \underline{u}_{j-1}$, então não há necessidade de calcular um evento cifrado, pois, por definição, este é igual a ε. Além disso, nesse caso, não há necessidade de atualizar $\underline{k}$, pois a chave não é utilizada, e também, o valor de $\underline{v}_j$ é atualizado para $\underline{v}_j = \underline{v}_{j-1}$ na linha 14. Assim, o processo de criptografia pode pular etapas e apenas aguardar a próxima leitura do vetor claro $\underline{u}_j$ na linha 2.

Algorithm 1: Processo de criptografia

```
1  Obter uma nova chave de criptografia k de K_S
2  Ler o vetor claro u_j
3  if j = 0 then
4      Transmitir v_0 ← u_0 ⊕ k
5      Retornar para linha 1
6  else if u_j ≠ u_{j-1} then
7      σ ← u_j ⊕ u_{j-1}
8      σ_c ← σ ⊕ k
9      if σ_c = 0 then
10         σ_c ← σ_μ ⊕ k
11     v_j ← v_{j-1} ⊕ σ_c e transmitir v_j
12     Retornar para linha 1
13 else
14     v_j ← v_{j-1} e transmitir v_j
15     Retornar para linha 2
```

O procedimento de decifração é apresentado no Algoritmo 2. Na linha 1, uma nova chave de criptografia $\underline{k}$ é calculada. Então, na linha 2, uma nova observação de $\underline{v}_j$ é aguardada. Se for o primeiro vetor cifrado observado, então $\underline{u}_0$ é calculado como $\underline{v}_0 \oplus \underline{k}$. Caso contrário, se $\underline{v}_j$ for diferente do vetor observado anteriormente $\underline{v}_{j-1}$, o evento cifrado $\underline{\sigma}_c$ é calculado na linha 7 e $\underline{\sigma}$ é calculado na lLinha 8. Se $\underline{\sigma}$ é igual a $\underline{\sigma}_\mu$, então na linha 10, é atribuído $\underline{k}$ a $\underline{\sigma}$. Então, na linha 11, o vetor claro $\underline{u}_j$ é calculado e o algoritmo retorna à linha 1. Note que, na linha 13, se $\underline{v}_j = \underline{v}_{j-1}$, não há necessidade de decifrar $\underline{v}_j$ já que nenhuma modificação ocorreu, ou seja, $\underline{u}_j$ é igual a $\underline{u}_{j-1}$.

Exemplo 2. Para ilustrar o procedimento de criptografia apresentado no Algoritmo 1, considere que é preciso crip-

Algorithm 2: Processo de decifração

```
1  Obter uma nova chave de criptografia k de K_S
2  Ler o vetor cifrado v_j
3  if j = 0 then
4  |   u_0 ← v_0 ⊕ k
5  |   Retornar para linha 1
6  else if v_j ≠ v_{j−1} then
7  |   σ_c ← v_j ⊕ v_{j−1}
8  |   σ ← σ_c ⊕ k
9  |   if σ = σ_μ then
10 |   |   σ ← k
11 |   u_j ← u_{j−1} ⊕ σ
12 |   Retornar para linha 1
13 else
14 |   u_j ← u_{j−1}
15 |   Retornar para linha 2
```

tografar e transmitir os três vetores claros do sistema dados por:

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

Na linha 1 do procedimento de encriptação, é necessário obter a primeira chave de criptografia $\underline{k}$ de K_S, o que requer a construção de parte da chave de fluxo K_S usando uma chave de 256 bits e um *nonce* de 96 bits, que foram omitidos por falta de espaço. Como cada matriz M_q, $q = 0, 1, \ldots, 2^{32} - 1$, fornece 512 bits de K_S, então, neste exemplo, para criptografar a mensagem composta pelos três vetores $\underline{u}_0$, $\underline{u}_1$ e $\underline{u}_2$, basta construir apenas a primeira matriz M_0. Após calcular M_0, pode-se obter, usando os primeiros 10 bits, a primeira chave de criptografia $\underline{k}$ dada por:

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

Após observar o primeiro vetor claro $\underline{u}_0$, na linha 4 do Algoritmo 1, o primeiro vetor cifrado $\underline{v}_0$, calculado como $\underline{v}_0 = \underline{u}_0 \oplus \underline{k} = [\,1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\,]^T$, é transmitido. Então, o Algoritmo 1 retorna à linha 1, onde uma nova chave $\underline{k}$ é obtida a partir de M_0 usando os próximos 10 bits dados por:

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

Após aguardar uma nova observação de um vetor claro, ou seja, o emissor lê o vetor $\underline{u}_1$, é verificado se este vetor é igual ao valor anterior $\underline{u}_0$. Note que, neste caso $\underline{u}_0 = \underline{u}_1$, e portanto o Algoritmo 1 vai para a linha 14, onde o vetor cifrado transmitido é uma cópia do vetor cifrado anterior, ou seja, $\underline{v}_1 = \underline{v}_0$. É importante ressaltar que neste caso o Algoritmo 1 vai para a linha 2 e, portanto, o valor do vetor chave $\underline{k}$ não é atualizado. Finalmente, quando o emissor observa o vetor claro $\underline{u}_2$, o evento $\underline{\sigma} = \underline{u}_1 \oplus \underline{u}_2 = [\,0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\,]^T$ é obtido. Na linha 8 do Algoritmo 1 é calculado o evento cifrado $\underline{\sigma}_c = \underline{\sigma} \oplus \underline{k} = [\,1\ 0\ 0\ 1\ 0\ 1\ 1\ 0\ 0\ 0\,]^T$. Note que $\underline{\sigma}_c$ é diferente de $\underline{0}$, portanto não há necessidade de usar o vetor $\underline{\sigma}_\mu$. Na linha 11 do Algoritmo 1, calcula-se o novo vetor cifrado $\underline{v}_2$, dado por $\underline{v}_2 = \underline{\sigma}_c \oplus \underline{v}_1 = [\,0\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 1\ 1\,]^T$. Finalmente, $\underline{v}_2$ é transmitido ao receptor e o processo de criptografia retorna à linha 1, onde um novo vetor $\underline{k}$ é obtido e o sistema aguarda uma nova leitura de um vetor claro.

Em resumo, os vetores cifrados transmitidos no canal de comunicação são:

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

É importante ressaltar que todos os vetores cifrados $\underline{v}_0$, $\underline{v}_1$ e $\underline{v}_2$ são completamente diferentes dos vetores claros do sistema e, portanto, um invasor não poderá recuperar adequadamente nenhuma informação do sistema observando os dados transmitidos.

Por outro lado, o receptor pode gerar os mesmos vetores chave $\underline{k}$ que o emissor e, portanto, o receptor pode calcular os vetores claros observando os vetores cifrados transmitidos usando o Algoritmo 2. Na linha 1 do Algoritmo 2, a primeira chave de criptografia $\underline{k} = [\,1\ 1\ 1\ 1\ 0\ 0\ 1\ 1\ 1\ 1\,]^T$ é calculada. Então, após observar o primeiro vetor cifrado $\underline{v}_0$, na linha 4 do Algoritmo 2, o receptor pode calcular $\underline{u}_0 = \underline{v}_0 \oplus \underline{k} = [\,0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 0\,]^T$. Então, $\underline{k}$ é atualizado na linha 1 para $\underline{k} = [\,1\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 0\,]^T$, e como $\underline{v}_1 = \underline{v}_0$, após observar $\underline{v}_1$, na linha 14, é visto que $\underline{u}_1 = \underline{u}_0$. Como $\underline{k}$ não foi utilizado, o Algoritmo 2 retorna à linha 2 para aguardar uma nova observação. Após observar $\underline{v}_2 \neq \underline{v}_1$, na linha 7, o evento cifrado $\underline{\sigma}_c = \underline{v}_1 \oplus \underline{v}_2$ é calculado, e então, na linha 8, $\underline{\sigma} = \underline{\sigma}_c \oplus \underline{k} = [\,0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\,]^T$ é recuperado. Finalmente, o vetor claro é calculado na linha 11 como $\underline{u}_2 = \underline{u}_1 \oplus \underline{\sigma} = [\,0\ 0\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 0\,]^T$. □

4. CONCLUSÕES

Neste artigo, é proposto um esquema de criptografia baseado em eventos para garantir a confidencialidade dos dados, evitando que um agente malicioso recupere as informações claras transmitidas entre emissor e receptor em uma rede de automação. Para tanto, é proposto o uso da cifra de fluxo ChaCha20, que é conhecida por ter alta resistência à criptoanálise. Os esquemas de criptografia e decifração não alteram o tamanho ou a estrutura dos dados de transmissão, o que leva a um método criptográfico eficiente para ser utilizado em redes de automação, pois insere um atraso relativamente pequeno na comunicação em relação a outros esquemas criptográficos. É apresentado um exemplo para ilustrar o esquema proposto de criptografia baseado em eventos.

REFERÊNCIAS

Alves, M.R., Pena, P.N., and Rudie, K. (2022). Discrete-event systems subject to unknown sensor attacks. *Discrete Event Dynamic Systems*, 32(1), 143–158.

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. Philadelphia, CA, USA.
Ji, Y., Wu, Y.C., and Lafortune, S. (2018). Enforcement of opacity by public and private insertion functions. *Automatica*, 93, 369–378.
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.
Li, X., Hadjicostis, C.N., and Li, Z. (2021). Extended insertion functions for opacity enforcement in discrete event systems. *IEEE Transactions on Automatic Control*, 1–1. 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*, 1–12.
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.
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.
Stallings, W. (2006). *Cryptography and network security, 4/E*. Pearson Education India.
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.

Apêndice A. CIFRA CHACHA20

A.1 Notação

Dados dois números positivos a e n, então a módulo n, $a\ mod(n)$, é o resto da divisão de a por n. Neste artigo, é usado o operador $\boxplus$, para denotar a operação adição módulo 2^{32}, ou seja, $a \boxplus b = (a+b)\ mod(2^{32})$.

A operação de rotação à esquerda para um número de 32 bits é denotada como $\lll$, ou seja, a operação de deslocar todos os bits de um número de 32 bits, um bit para a esquerda, e colocar o bit mais à esquerda (mais significativo) como o bit mais à direita do número de 32 bits. Também é definida a operação de rotação à esquerda para um determinado número $\ell \in \mathbb{N}$, tal que dado um número de 32 bits a então $a \lll \ell$ representa a repetição da operação de rotação à esquerda de a, ℓ vezes.

A.2 Geração do keystream

É apresentado no Algoritmo 3 a função Quarter-round que é usada durante a geração do *keystream* na criptografia ChaCha20 (Bernstein, 2008a).

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

Input: números de 32 bits a, b, c, d
Output: números de 32 bits modificados 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 $b \leftarrow b \lll 7$

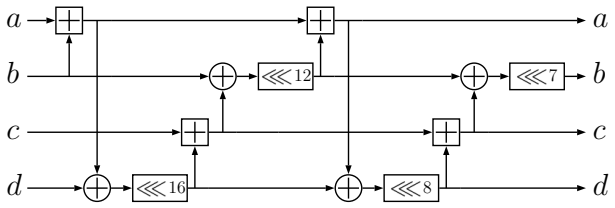

Figura A.1. Função Quarter-round.

Para ilustrar a função **Quarter-round**, a operação sobre as entradas a, b, c e d está representada na Figura A.1. Observe que, os valores originais de a, b, c e d são atualizados pela função **Quarter-round** fazendo operações de rotação à esquerda, XOR e adição módulo 2^{32}.

As matrizes M_q, $q = 0, \ldots, 2^{32}-1$, usadas para gerar o *keystream* K_S, são calculadas de acordo com o Algoritmo 4. Para tanto, na linha 1, S é construído como:

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

em que $const_0$, $const_1$, $const_2$, e $const_3$ são números binários de 32 bits apresentados em Bernstein (2008a). Os valores de $k_{s_0}, k_{s_1}, k_{s_2}, k_{s_3}, k_{s_4}, k_{s_5}, k_{s_6}$ e k_{s_7} são números binários com 32 bits obtidos após o particionamento da chave secreta k_s, que possui 256 bits, em 8 partes, e *count* é um contador de 32 bits. Os valores de n_0, n_1 e n_2 são números binários com 32 bits obtidos após o

particionamento do *nonce* η, que possui 96 bits, em três partes.

Na linha 2 do Algoritmo 4, é feito uma cópia da matriz S, denotada por S_0. Para facilitar a notação, cada número binário de 32 bits da matriz S_0 é apresentado da seguinte forma:

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

Após definir a matriz S_0, esta é modificada, gerando uma nova matriz, seguindo as operações das linhas 5 a 12. As operações:

$$\begin{aligned} [s_{0,0}, s_{1,0}, s_{2,0}, s_{3,0}] &= \textbf{Quarter-round}(s_{0,0}, s_{1,0}, s_{2,0}, s_{3,0}) \\ [s_{0,1}, s_{1,1}, s_{2,1}, s_{3,1}] &= \textbf{Quarter-round}(s_{0,1}, s_{1,1}, s_{2,1}, s_{3,1}) \\ [s_{0,2}, s_{1,2}, s_{2,2}, s_{3,2}] &= \textbf{Quarter-round}(s_{0,2}, s_{1,2}, s_{2,2}, s_{3,2}) \\ [s_{0,3}, s_{1,3}, s_{2,3}, s_{3,3}] &= \textbf{Quarter-round}(s_{0,3}, s_{1,3}, s_{2,3}, s_{3,3}) \end{aligned}$$

constituem uma rodada de colunas, e as operações:

$$\begin{aligned} [s_{0,0}, s_{1,1}, s_{2,2}, s_{3,3}] &= \textbf{Quarter-round}(s_{0,0}, s_{1,1}, s_{2,2}, s_{3,3}) \\ [s_{0,1}, s_{1,2}, s_{2,3}, s_{3,0}] &= \textbf{Quarter-round}(s_{0,1}, s_{1,2}, s_{2,3}, s_{3,0}) \\ [s_{0,2}, s_{1,3}, s_{2,0}, s_{3,1}] &= \textbf{Quarter-round}(s_{0,2}, s_{1,3}, s_{2,0}, s_{3,1}) \\ [s_{0,3}, s_{1,0}, s_{2,1}, s_{3,2}] &= \textbf{Quarter-round}(s_{0,3}, s_{1,0}, s_{2,1}, s_{3,2}) \end{aligned}$$

constituem uma rodada diagonal.

O processo de realizar uma rodada de coluna seguida de uma rodada diagonal é repetido 10 vezes no loop da linha 4. Após isso, a matriz M_q é formada na linha 14.

É importante ressaltar que o Algoritmo 4 produz uma matriz M_q de 512 bits como saída. Esse processo pode ser repetido para cada valor diferente do contador *count* para gerar uma nova matriz M_q de 512 bits que pode ser calculada usando a mesma chave secreta k_s e *nonce* η. Assim, o número de bits da parte do *keystream* K_S que é desejado calcular, fornece o número de vezes que o Algoritmo 4 deve ser executado gerando diferentes matrizes M_q. Após obter a matriz M_q, parte do *keystream* K_S é obtido convertendo as linhas de M_q em um único número binário com 512 bits, concatenando as linhas lado a lado.

Algorithm 4: Cálculo da matriz M_q

Input: $k_s = k_{s_0} \dots k_{s_7}$, $\eta = n_0 n_1 n_2$, *count*, $const_0$, $const_1$, $const_2$, $const_3$

Output: M_q, onde q é a representação decimal de *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 $\quad [s_{0,0}, s_{1,0}, s_{2,0}, s_{3,0}] = \textbf{Quarter-round}(s_{0,0}, s_{1,0}, s_{2,0}, s_{3,0})$

6 $\quad [s_{0,1}, s_{1,1}, s_{2,1}, s_{3,1}] = \textbf{Quarter-round}(s_{0,1}, s_{1,1}, s_{2,1}, s_{3,1})$

7 $\quad [s_{0,2}, s_{1,2}, s_{2,2}, s_{3,2}] = \textbf{Quarter-round}(s_{0,2}, s_{1,2}, s_{2,2}, s_{3,2})$

8 $\quad [s_{0,3}, s_{1,3}, s_{2,3}, s_{3,3}] = \textbf{Quarter-round}(s_{0,3}, s_{1,3}, s_{2,3}, s_{3,3})$

9 $\quad [s_{0,0}, s_{1,1}, s_{2,2}, s_{3,3}] = \textbf{Quarter-round}(s_{0,0}, s_{1,1}, s_{2,2}, s_{3,3})$

10 $\quad [s_{0,1}, s_{1,2}, s_{2,3}, s_{3,0}] = \textbf{Quarter-round}(s_{0,1}, s_{1,2}, s_{2,3}, s_{3,0})$

11 $\quad [s_{0,2}, s_{1,3}, s_{2,0}, s_{3,1}] = \textbf{Quarter-round}(s_{0,2}, s_{1,3}, s_{2,0}, s_{3,1})$

12 $\quad [s_{0,3}, s_{1,0}, s_{2,1}, s_{3,2}] = \textbf{Quarter-round}(s_{0,3}, s_{1,0}, s_{2,1}, s_{3,2})$

13 $\quad h \leftarrow h + 1$

14 $M_q = S \boxplus \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix}$