Esquema de Estruturação SbC-EC para Log Seguro

Sérgio Câmara[1,2], **Luci Pirmez**[1], **Luiz F.R.C. Carmo**[1,2]

[1]Programa de Pós-Graduação em Informática - Instituto Tércio Pacitti / IM
Universidade Federal do Rio de Janeiro, 21.941-901, RJ – Brasil

[2]Instituto Nacional de Metrologia, Qualidade e Tecnologia (Inmetro)
Av. Nossa Senhora das Graças, 50, Xerém, Duque de Caxias, 25250-020, RJ – Brasil

{smcamara,lfrust}@inmetro.gov.br, luci@nce.ufrj.br

Abstract. *Secure log schemes ensure detection of possible attacks against audit logs located in devices under an unprotected environment. This paper describes the structure scheme SbC-EC for secure logging, suitable for storage and network communication constrained devices, presenting two new features: Split by Category and Entry Compaction. We also describe the SbC-EC MAC secure log scheme, which implements the new proposed structure along with symmetric cryptography primitives and the FssAgg authentication scheme for protecting the log files. SbC-EC MAC presents a storage gain in comparison to other existing symmetric secure log schemes.*

Resumo. *Esquemas de log seguro garantem a detecção de possíveis ataques contra o log de auditoria residente em dispositivos de um ambiente não protegido. Este artigo descreve o esquema de estruturação SbC-EC para log seguro, apropriado para dispositivos com restrições de armazenamento e de comunicação em rede, apresentando duas novas características: Separação por Categoria e Compactação de Entradas. Descrevemos também o esquema de log seguro SbC-EC MAC, que implementa a nova estruturação proposta em conjunto com primitivas da criptografia simétrica e o esquema de autenticação FssAgg para a proteção dos arquivos de log. O SbC-EC MAC apresenta um ganho de armazenamento em relação a outros esquemas simétricos de log seguro.*

1. Introdução

O log de auditoria é um importante mecanismo da computação forense, capaz de oferecer segurança e confiabilidade aos sistemas computacionais. No log de auditoria podem ser registrados, por exemplo, eventos críticos como o estado e erros de execução de programas, atividades de rede e de usuários e modificação de dados. Dessa forma, um arquivo de log pode ser utilizado para efetuar a análise e diagnóstico de incidentes de segurança, permitindo a rastreabilidade da causa raiz de um problema, ou ataque, no sistema.

Devido aos benefícios obtidos com a implementação do log de auditoria e por ser de grande valia para uma análise forense, os arquivos de log são constantes alvos de ataques após a invasão de um sistema. Um atacante experiente espera apagar, ou modificar, qualquer informação registrada capaz de revelar sua identidade ou permitir a detecção de traços do comprometimento do sistema. Além disso, o atacante espera

*Artigo financiado com recursos do projeto Pronametro 52600.011757/2014.

manter os métodos de ataque transparentes para o administrador do sistema, escondendo as brechas de segurança para serem utilizadas em um futuro ataque [Bellare e Yee 1997]. Dessa forma, torna-se essencial manter o log de auditoria íntegro e seguro contra qualquer tipo de manipulação maliciosa. Uma solução possível para garantir a proteção ao log de auditoria inclui o uso de técnicas de hardware que são resistentes à violação (*tamper-proof*). Entretanto, os componentes de hardware demandados por essas técnicas podem onerar muito o custo de produção de um dispositivo com restrições de recursos como memória e processamento (p. ex., sensores sem-fio) [Ma e Tsudik 2007]. Outra técnica alternativa é a adoção de armazenamentos controlados por software do tipo WORM, *Write Once Read Multiple*, que garantem que uma informação uma vez escrita não pode ser modificada posteriormente. No entanto, esses armazenamentos são vulneráveis às ações maliciosas de atacantes internos com alto grau de privilégio e acesso físico aos discos [Oprea e Bowers 2009].

Outra técnica considerada para garantir a proteção ao log de auditoria é a utilização de um servidor *online* de coleta, responsável por requisitar e armazenar em tempo real as entradas de log dos dispositivos. Essa abordagem possui algumas desvantagens como a dependência de uma comunicação sempre ativa entre os dispositivos e o servidor central de coleta, o que pode não ser factível em determinados tipos de cenários. Com isso, a comunicação incerta, ou não frequente, com o servidor de coleta, além do armazenamento local por tempo indeterminado das entradas de log nos dispositivos, são condições que precisam ser consideradas. Ao longo do tempo, alguns protocolos e técnicas criptográficas foram criados e aprimorados a fim de garantir a proteção dos arquivos de log nos dispositivos, desenvolvendo, portanto, os mecanismos de logs seguros. Os logs seguros são mecanismos de segurança que possuem propriedades como *forward-security*[1] , ser não falsificável, ser a prova de violação, oferecer verificação de integridade (pública ou não), entre outras adicionais como prover confidencialidade e controle de acesso aos registros, verificação individual de entradas e busca por palavras-chave.

Juntamente com a necessidade de segurança, os arquivos de log dos dispositivos tendem a crescer indefinidamente com o passar do tempo, podendo conter milhares de entradas de logs, necessitando cada vez mais espaço de armazenamento [Ma e Tsudik 2009b]. Além disso, as boas práticas do padrão *ISA - Security for industrial automation and control systems*, antiga ISA-99, determinam, como requisito obrigatório, que sistemas embarcados devem oferecer uma capacidade de armazenamento suficiente para registrar os eventos auditáveis, a fim de que não haja perda de informações entre uma auditoria e outra [International Electrotechnical Commission 2011]. Logo, em um cenário onde os dispositivos apresentam restrições de memória e de comunicação em rede, as condições de um arquivo de log sempre crescente e o requisito de um armazenamento suficiente conflitam diretamente.

Com base nos problemas de segurança e armazenamento de arquivos de log citados, o presente trabalho descreve um novo esquema de estruturação de log seguro apropriado para dispositivos com restrições de memória de armazenamento e de comunicação em rede (p. ex., em aplicações de sistemas ciber-físicos, sistemas embarcados, etc). O esquema de estruturação SbC-EC apresenta duas características com o objetivo de reduzir

[1]Propriedade que assegura o não comprometimento do uso de chaves utilizadas no passado a partir da revelação de qualquer informação no momento presente [Bellare e Yee 2003].

o espaço necessário para armazenar entradas de log: (i) Separação por Categoria (*Split by Category*, SbC) e (ii) Compactação de Entradas de log (*Entry Compaction*, EC). Além disso, apresentamos um novo esquema de log seguro que utiliza essa nova estruturação, o esquema SbC-EC MAC. O SbC-EC MAC utiliza o esquema de autenticação FssAgg em conjunto com outros mecanismos para estabelecer a segurança necessária ao log de auditoria. Além disso, avaliamos o SbC-EC MAC juntamente aos esquemas de Schneier-Kelsey [Schneier e Kelsey 1998] e FssAgg MAC [Ma e Tsudik 2007], a fim de mostrar que ele permite um maior ganho de espaço de armazenamento em relação a esses dois.

O restante do artigo está organizado da seguinte maneira. A seção 2 destaca os trabalhos importantes na área de logs seguros, mencionando as principais características dos esquemas e possíveis desvantagens. A seção 3 descreve o modelo geral do sistema adotado no trabalho e o modelo de atacante, detalhando seus modos de operação. Na seção 4 é descrita a proposta do novo esquema de estruturação SbC-EC para log seguro e suas novas características. Em seguida, apresentamos um novo esquema de log seguro, o SbC-EC MAC. Cada uma de suas funções e os passos detalhados de funcionamento são descritos na seção 5. A seção 6 apresenta uma discussão sobre a segurança do SbC-EC MAC, incluindo suas propriedades e possíveis vulnerabilidades. Na seção 7, realizamos uma avaliação comparativa entre o nosso esquema de log e outros dois esquemas de log seguro da literatura e, por fim, concluímos nosso trabalho e indicamos nossos esforços futuros na seção 8.

2. Trabalhos Relacionados

A seguir, destacamos os principais trabalhos relacionados à elaboração de esquemas e características de logs seguros. Diferentemente da abordagem na qual os dispositivos possuem comunicação com um servidor *online* 100% do tempo, estes trabalhos tem como foco os esquemas para proteção dos logs "em repouso" (*logs at rest*). Esses esquemas oferecem soluções tanto no domínio da criptografia simétrica quanto na de chave-pública. Os trabalhos de Bellare e Yee foram os primeiros a incorporar as características de *forward security* aos logs de auditoria a fim de garantir posteriormente a integridade do fluxo das entradas de log (*forward-secure stream integrity*). Eles montam seu esquema utilizando MACs (*Message authentication codes*), onde as entradas de log são indexadas de acordo com períodos de tempo [Bellare e Yee 1997] [Bellare e Yee 2003].

O esquema Schneier-Kelsey descreve um esquema de chave simétrica utilizando cadeia de hash autenticada por MACs para estabelecer uma dependência entre as entradas de log, permitindo detectar qualquer alteração feita na ordem das entradas. O trabalho também descreve uma implementação do protocolo para sistemas distribuídos, incluindo troca de mensagens, criação e fechamento de arquivo de log [Schneier e Kelsey 1998] [Schneier e Kelsey 1999]. Uma das desvantagens do esquema Schneier-Kelsey é a sobrecarga na comunicação e no armazenamento das entradas de log, uma vez que para cada entrada é guardado um MAC para autenticação individual. Além disso, o esquema é suscetível ao ataque de *truncation*, no qual o atacante é capaz de eliminar *n* últimas entradas do log sem que essa manobra seja percebida em uma verificação posterior.

O Logcrypt é uma extensão do esquema de Schneier-Kelsey utilizando criptografia de chave-pública [Holt 2006]. Da mesma forma, esse esquema apresenta uma grande sobrecarga na comunicação, por estabelecer frequentes trocas de pares de cha-

ves entre os dispositivos e o servidor central, e no armazenamento, em consequência das assinaturas digitais para uma, ou mais, entradas de log. O Logcrypt também é vulnerável ao ataque de *truncation*. O trabalho de Accorsi descreve o BBox, uma caixa-preta para armazenamento do log de auditoria em sistemas distribuídos com apoio de componentes de computação confiável, como o TPM (*Trusted Plataform Module*). O esquema de log seguro que compõe o BBox segue a mesma linha do esquema de Schneier-Kelsey, utilizando cadeias de hash assinadas digitalmente [Accorsi 2011] [Accorsi 2013]. Entretanto, essa abordagem também oferece desvantagens em relação à sobrecarga de comunicação/armazenamento devido às assinaturas de cada entrada de log. Além disso, o modelo centralizador adotado não oferece segurança apropriada aos logs nos dispositivos periféricos, os quais podem ser comprometidos de maneiras não previstas no estudo.

Ma e Tsudik desenvolveram o primeiro esquema de autenticação de log seguro apropriado para dispositivos com restrições de comunicação e armazenamento. O esquema de autenticação *FssAgg* (*Forward-Secure Sequential Aggregate*), proposto tanto para a criptografia convencional quanto para a de chave-pública, permite que o dispositivo assinante combine diferentes tags de autenticação criadas em diferentes chaves/períodos de tempo em uma única tag de tamanho constante. Dessa forma, a autenticação do arquivo de log depende de apenas uma tag, resistente ao ataque de *truncation* devido à propriedade "all-or-nothing" de verificação de assinatura [Ma e Tsudik 2007] [Ma e Tsudik 2008] [Ma e Tsudik 2009b]. O esquema SbC-EC MAC, apresentado no presente trabalho, utiliza a autenticação *FssAgg* a fim de compor a tag única para o verificador.

Yavuz et al. desenvolveram o esquema BAF (*Blind Aggregation Forward*), aplicando também a autenticação *FssAgg*. O BAF oferece maneiras menos custosas de realizar a assinatura digital das entradas de log, sendo, portanto, uma opção para dispositivos com baixo poder computacional [Yavuz e Ning 2009] [Yavuz et al. 2012]. Em particular, o nosso trabalho descreve esquemas para sistemas com restrições de em espaço de armazenamento e banda de rede. Dessa forma, optamos por um esquema de autenticação baseado em criptografia simétrica por se mostrar mais adequada aos nossos propósitos.

O presente trabalho se diferencia dos demais na forma de como estruturar o log seguro usado pelos dispositivos. Enquanto os outros trabalhos organizam as entradas de log de acordo com o tempo em que foram geradas e em uma única sequência, nossa estruturação propõe uma organização pelo tempo da geração das entradas porém separadas em categorias de tipos de log. Além disso, essa divisão permite que a verificação do log de auditoria seja feita em subconjuntos de entradas de log, ou seja, por uma categoria de cada vez. Isso oferece, portanto, uma opção intermediária de verificação entre a verificação integral do arquivo de log, quando não necessário, e a verificação individual de entradas (p. ex., *immutability* [Ma e Tsudik 2009b]), onde a última introduz uma sobrecarga de armazenamento para cada uma das entradas.

3. Modelos

Nesta seção, é apresentada a descrição do Modelo do Sistema (seção 3.1) e do Atacante (seção 3.2) utilizados neste trabalho.

3.1. Modelo do Sistema

O modelo de sistema utilizado nesse trabalho é baseado no modelo descrito por Schneier [Schneier e Kelsey 1999]. O sistema é composto basicamente por três partes, T (máquina

confiável), U (máquina não confiável) e V (verificador semi-confiável), que são detalhadas em seguida:

1. T, a máquina confiável. T pode ser instanciado de diversas maneiras, p. ex., como um servidor em um ambiente controlado. T possui armazenamento suficiente para guardar todos os logs de U e autoriza um verificador V a ter acesso aos logs de U. Por fim, T realiza a verificação final dos arquivos de log, uma vez transmitidas a ele.
2. U, a máquina não confiável. Não há garantias de que essa máquina está totalmente segura contra atacantes, erros de software, entre outras ameaças. Nela são gerados e armazenados os logs por um período de tempo indeterminado. Uma vez comprometido, U agirá da forma que o atacante estipular. U é capaz de se comunicar e interagir com T e V.
3. V, o verificador semi-confiável. O verificador pode realizar a função de inspecionar a integridade do arquivo de log de auditoria, porém sem alterar nenhuma entrada. Em geral, V se caracteriza por um pequeno grupo de entidades autorizadas (p. ex., administradores de sistemas) a obter e ler uma cópia dos logs de U, quando necessário.

Durante o período de inicialização do sistema é realizada uma comunicação inicial entre o servidor T e o dispositivo U a fim de compartilhar os segredos iniciais do esquema de log seguro. Assumimos que todas as transmissões entre as partes do sistema sejam estabelecidas por uma comunicação segura, utilizando protocolos reconhecidos de autenticação e de troca-de-chaves.

O dispositivo U gera entradas de log de formato bem definido e as guarda em seu armazenamento não volátil. Em intervalos de tempo não frequentes e não determinísticos, o dispositivo U transmite seus logs armazenados para o servidor T, que, por sua vez, verifica a integridade das informações enviadas e as guarda de forma segura. Além da capacidade de U de armazenar logs, ele também é capaz de apagar informações da memória a fim de cumprir os protocolos de autenticação. Assumimos, também, que U é capaz de gerar números pseudo-aleatórios de maneira robusta.

Abaixo apresentamos as primitivas criptográficas e suas notações usadas ao longo do artigo:

1. $prf(K_0)$ é uma *pseudo-random function*, capaz de transformar a chave K_0 para K_1.
2. $\mathcal{H}(X)$ é uma *one-way function* que recebe uma mensagem X e calcula um resumo de tamanho fixo. $\mathcal{H}$ pode ser implementado com SHA-1 [National Institute of Standards and Technology 2012].
3. $E_{K_0}(X)$ é a cifragem simétrica de X usando a chave K_0. E pode ser um algoritmo como o AES [National Institute of Standards and Technology 2001].
4. $MAC_{K_0}(X)$ é o *message authentication code* simétrico de X usando a chave K_0. Pode ser implementado com HMAC [Bellare et al. 1996].

3.2. Modelo de Atacante

Como dito anteriormente, os logs de auditoria são alvos importantes para atacantes que queiram esconder traços de seu ataque, agindo de forma a apagar, modificar, inserir ou reordenar as entradas de log a fim de atingir seu objetivo. De fato, os esquemas de log

seguro não protegem contra ações maliciosas no arquivo de log. A característica principal de um esquema de log seguro é garantir que qualquer modificação indevida realizada seja detectada no ato da verificação do arquivo de log.

A seguir, consideramos as possibilidades de um atacante durante o comprometimento de U. Dada a última transmissão de entradas de log de U para T no instante de tempo t_1 e que o comprometimento de U acontece no instante de tempo t_2, é assumido que:

- O atacante toma conhecimento das chaves A_{t_2} e B_{t_2}, guardadas em U, no momento do comprometimento e é capaz de derivar todas as evoluções futuras dessas chaves.
- O atacante é capaz de gerar as tags do verificador e da parte confiável, uma vez que o algoritmo de geração de tags é público.
- As entradas de log que foram geradas em U, após o instante de tempo t_1, não podem ser consideradas confiáveis. O atacante tem a capacidade de apagá-las, modificá-las, reordená-las ou inserir novas entradas.
- O atacante desconhece as chaves geradoras das entradas de log $L_{t_1}...L_{t_2-1}$. Portanto, qualquer manipulação realizada nessas entradas poderá ser detectada em uma futura verificação do log por V ou T.
- Qualquer entrada de log gerada após L_{t_2}, inclusive, poderá ser manipulada a fim de contornar uma detecção posterior.

4. Esquema de Estruturação SbC-EC para Log Seguro

Um dos desafios relacionados à geração de logs de auditoria em dispositivos em geral é a necessidade de armazenamento de um grande, e crescente, volume de informação ao longo do tempo de operação. Com o objetivo de diminuir o espaço necessário para o armazenamento dessas informações, nesse trabalho é proposto um novo esquema de Estruturação para log seguro, denominado SbC-EC, que apresenta uma nova maneira de organizar semanticamente as entradas de log.

A estratégia principal adotada no esquema de estruturação SbC-EC reside na eliminação de informação redundante encontrada nos *payloads* das entradas de log semelhantes. Uma vez que o mecanismo de log seguro detém a propriedade de *forward-integrity*, é impossível realizar alterações nas entradas passadas de forma manter a integridade do fluxo de entradas, logo, qualquer alteração de *payload* deve ser executada na entrada antes de ser logada. Para eliminar informações redundantes do *payload*, a entrada a ser logada L_i deverá referenciar alguma anterior semelhante a ela, porém, isso torna-se impeditivo quando o número de entradas é muito grande, necessitando realizar buscas cada vez maiores para logar uma só entrada. Dessa forma, limitamos a comparação apenas a um nível, ou seja, caso o *payload* de L_i for semelhante ao de L_{i-1}, L_i é logada de forma compactada. Essa manobra estabelece a propriedade de *Compactação de Entradas* (*Entry Compaction*, EC) nesse esquema de estruturação.

A *Compactação de Entradas* funciona da seguinte maneira. Consideramos um *payload* bem estruturado P_n, composto pela tupla $(ID_n, M_n, [Params_n], TS_n)$, onde ID_n é o identificador único da mensagem de log, M_n é a mensagem de log, $[Params_n]$ é o conjunto de parâmetros referente à mensagem de log, caso haja, e TS_n é o *timestamp* da geração do *payload*. Quando um novo payload P_i está para ser logado, é verificado se $ID_i = ID_{i-1}$, caso seja verdade, o *payload* P_i será composto pela tupla

$(TS_{dif}, [Params_i])$, onde TS_{dif} é a diferença horária entre o *timestamp* de P_i e o *timestamp* de P_{i-1} ($TS_{dif} = TS_i - TS_{i-1}$).

Uma vez que os eventos auditáveis são de diversos tipos, é baixa a probabilidade de que, em um único fluxo de entradas, duas entradas semelhantes sejam logadas consecutivamente. A partir disso, desejamos aumentar essa probabilidade e, consequentemente, compactar mais entradas e poupar mais espaço de armazenamento. Para que isso seja possível, a característica de *Separação por Categoria* (*Split by Category*, SbC) foi estabelecida. A estruturação SbC-EC classifica cada entrada de log gerada em uma das $1..c$ categorias pré-estabelecidas e a concatena junto às demais da mesma categoria, em diferentes fluxos de entradas de log. Cada fluxo corresponde a um arquivo de log.

Assumimos que um componente de software saiba identificar qual a categoria de cada *payload* de log antes de ser concatenada como uma entrada no arquivo de log. Uma categoria de log corresponde aos grupos de registros de eventos auditáveis semelhantes, tais como eventos de controle de acesso, erros de requisição, eventos de sistema, alteração de configuração, potenciais atividades de reconhecimento e eventos de log [International Electrotechnical Commission 2011].

5. Esquema de Log Seguro SbC-EC MAC

Nessa seção é apresentado um novo esquema de log seguro, o SbC-EC MAC, utilizando criptografia simétrica e com base na estruturação SbC-EC. A criptografia simétrica é aqui aplicada pois oferece uma sobrecarga de autenticação menor do que a criptografia assimétrica em relação à quantidade de dados necessários. Além disso, o SbC-EC MAC utiliza o esquema simétrico de autenticação *FssAgg* [Ma e Tsudik 2009b] adaptado à nova estruturação, a fim de gerar as tags de autenticação necessárias das entradas de log. O esquema *FssAgg* estipula uma tag única (assinatura agregada) a fim de autenticar todas as entradas de log existentes em um arquivo de log.

A seguir, são descritos os algoritmos presentes no esquema SbC-EC MAC:

- *SbC-EC.Skg – Secret key generation*, o algoritmo de geração de chave secreta é usado para gerar uma chave inicial de tamanho seguro a ser usada pelo algoritmo de criptografia simétrica.
- *SbC-EC.Ckg – Category key generation*, o algoritmo de geração das chaves iniciais de verificador para cada categoria. O algoritmo recebe como entrada uma chave inicial de verificador A_0 e o número de categorias c e responde, como saída, as chaves de verificador $A_{1,1}, A_{1,2}, ..., A_{1,c}$, uma para cada categoria.
- *SbC-EC.Vsig – Verifier signature*, o algoritmo assina-e-agrega (*sign-and-aggregate*) a fim de formar a tag de autenticação do verificador. O algoritmo recebe como entradas uma chave secreta, uma mensagem a ser assinada e a tag de autenticação computada até aquele ponto. Ele calcula a assinatura da mensagem e a agrega à tag já existente. Ao final, o algoritmo realiza uma atualização da chave secreta usada. O *SbC-EC.Kupd*, *Key update*, é realizado dentro do algoritmo de assinatura por questões de segurança. Esses algoritmos remetem aos *FssAgg.Asig* e *FssAgg.Upd* descritos por Ma e Tsudik.
- *SbC-EC.Tsig – Trusted party signature*, o algoritmo que gera a tag de autenticação da parte confiável. Ele recebe como entradas uma chave secreta e c tags de autenticação, $c \geq 1$. O algoritmo concatena e assina as tags da entrada para

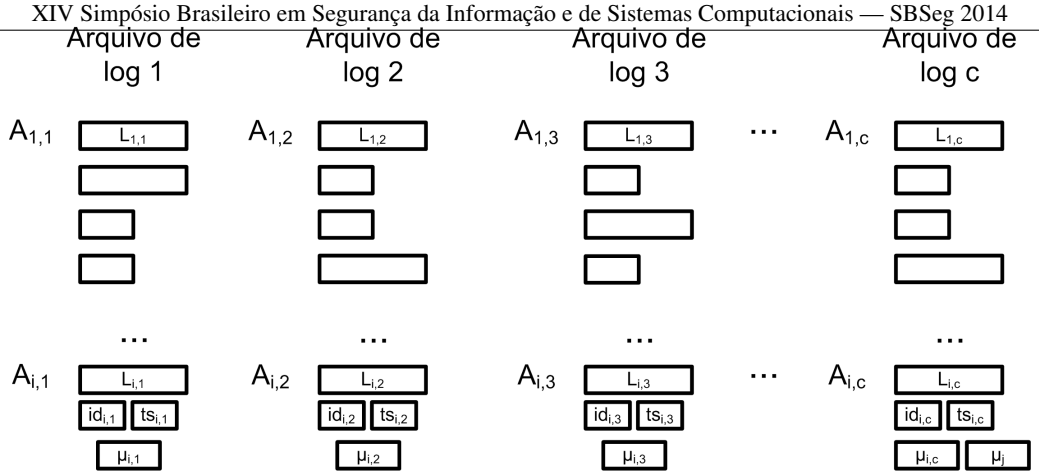

Figura 1. Representação dos elementos do SbC-EC MAC.

gerar uma única tag. Por fim, o algoritmo atualiza a chave secreta, utilizando *SbC-EC.Kupd*, por questões de segurança.

- *SbC-EC.Vver – Verifier verification*, o algoritmo de verificação da parte verificadora V recebe uma chave secreta, um fluxo de entradas de log e sua tag de autenticação e responde se a sequência de entradas está íntegra ou não. Esse algoritmo remete ao *FssAgg.Aver*.
- *SbC-EC.Tver – Trusted party verification*, o algoritmo de verificação da parte confiável recebe uma chave secreta, c fluxos de entradas de log em conjunto com suas c tags de autenticação, $c \geq 1$, e uma tag de autenticação da parte confiável. O algoritmo recalcula a tag da parte confiável a partir das entradas e responde, caso essa tag for igual à tag da parte confiável informada, os fluxos de log estão íntegros, caso contrário, não.

Um dos mecanismos usados pelo SbC-EC MAC é o de evolução de chaves secretas, usada também em outros esquemas de log seguro e em aplicações que exigem propriedades como *forward-security*. Esse mecanismo permite que uma chave K_i evolua para K_{i+1} de forma determinística através de uma função pseudo-randômica $prf(K_i)$. É impossível obter K_i a partir de K_{i+1}.

No SbC-EC MAC temos que, para cada arquivo de log representando uma categoria m, onde $1 \leq m \leq c$, existe uma tag única de verificador. Essa tag de verificador é representada por $\mu_{i,m}$, correspondente à i-ésima entrada de log da categoria m. Existe, também, uma única tag da parte confiável μ_j, correspondente à j-ésima evolução da chave secreta da parte confiável B_j. A Figura 1 mostra uma representação dos diferentes arquivos de log, cada um com suas correspondentes entradas de log, compactadas ou não, as chaves secretas de evolução e tags de verificador, e a única tag da parte confiável existente no esquema.

A seguir, descrevemos os passos do esquema SbC-EC MAC desde a preparação e inicialização dos arquivos de log, a concatenação e compactação de novas entradas de log e, por fim, as maneiras de realizar as verificações do log de auditoria para as diferentes partes do sistema.

5.1. Inicialização do log de auditoria

O dispositivo U deve efetuar alguns passos de preparação do arquivo de log antes de criá-lo. Primeiramente, U deve utilizar o algoritmo *SbC-EC.Skg* para gerar duas chaves

secretas: (i) A_0, chave inicial do verificador V e (ii) B_0, chave inicial da parte confiável T. U deve definir c, o número de categorias de logs, podendo esse dado estar predefinido e protegido no dispositivo. A partir disso, U deve informar a T esses 3 dados (A_0, B_0, c). Assumimos que a comunicação entre U e T seja segura, autenticada e à prova de ataques.

Após a confirmação de que T recebeu os dados enviados, U utiliza a função *SbC-EC.Ckg* para gerar todas as chaves iniciais de verificador de cada categoria. A partir disso, inicializa-se c arquivos de log, um para cada categoria. Para garantir a detecção de um ataque de deleção total do arquivo de log, antes de qualquer entrada legítima ser guardada, uma entrada "*dummy*" é gerada e concatenada em todos os arquivos. Essa medida evita que, após um atacante comprometer U e apagar todas as entradas de log existentes até o momento, ele não consiga esconder esse fato reivindicando que o arquivo de log nunca foi inicializado.

5.2. Concatenação e compactação de entradas de log

Para cada última entrada de cada arquivo de log, são guardados em claro (ou seja, sem estar criptografados) os dados $ID_{i-1,m}$ e $TS_{i-1,m}$, isso se justifica pelo processo de compactação de entradas, como explicado na seção 4. Descrevemos, a seguir, os procedimentos necessários durante o procedimento de concatenação de uma nova entrada de log. Primeiramente, é descrita a concatenação de uma entrada normal (quando $ID_i \neq ID_{i-1}$) e, em seguida, a concatenação de uma entrada compactada (quando $ID_i = ID_{i-1}$).

A partir da geração de um novo *payload* P, uma função decisora de categoria será responsável por escolher qual arquivo de log m deve concatenar a nova entrada. Uma vez definida a categoria, inicia-se o procedimento de concatenação da entrada. Assumimos que o arquivo de log da categoria m já possui as entradas $L_{1,m}$, $L_{2,m}$, $L_{3,m}$, ..., $L_{i-1,m}$. Os passos necessários para concatenar a entrada $L_{i,m}$ não compactada são descritos a seguir (Figura 2):

1. ID_i é comparado com ID_{i-1}. Como são diferentes nesse caso, ID_i é guardado e ID_{i-1} é apagado.
2. TS_i é guardado e TS_{i-1} é apagado.
3. $L_{i,m} = E_{A_{i,m}}(P_i)$, gera a nova entrada a partir da cifragem simétrica do *payload* P_i com a chave $A_{i,m}$.
4. Gera a i-ésima assinatura agregada $\mu_{i,m}$, tag do verificador, calculada conforme a Equação 1 [Ma e Tsudik 2009b]:

$$\mu_{i,m} = \mathcal{H}(\mathcal{H}(...\mathcal{H}(\mu_{1,m}||MAC_{A_{1,m}}(L_{1,m}))...)||MAC_{A_{i,m}}(L_{i,m})) \quad (1)$$

5. *SbC-EC.Kupd($A_{i,m}$)*, evolui a chave $A_{i,m}$ para $A_{i+1,m}$.
6. Gera a j-ésima tag da parte confiável, calculada conforme a Equação 2:

$$\mu_j = MAC_{B_j}(\mu_1||\mu_2||...||\mu_c) \quad (2)$$

   onde μ_m é a atual tag de verificador da categoria m, $1 \leq m \leq c$.
7. *SbC-EC.Kupd(B_j)*, evolui a chave B_j para B_{j+1}.

Para a concatenação de uma nova entrada de log $L_{i,m}$ em que ID_i é igual ao da última entrada $L_{i-1,m}$, a nova entrada será compactada. Os passos necessários para concatenar $L_{i,m}$ compactada são descritos abaixo:

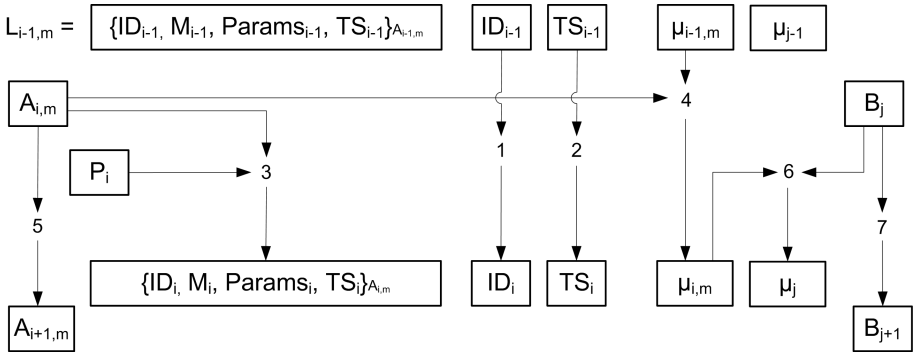

Figura 2. Concatenação de uma nova entrada $L_{i,m}$ não compactada.

1. ID_i é comparado com ID_{i-1}. Como são iguais nesse caso, ID_i é descartado e ID_{i-1} é mantido.
2. $TS_{dif} = TS_i - TS_{i-1}$, a diferença horária entre as entradas. TS_i é guardado e TS_{i-1} é apagado.
3. $L_{i,m} = E_{A_{i,m}}(P_i)$, gera a nova entrada a partir da cifragem simétrica do *payload* P_i com a chave $A_{i,m}$. Onde $P_i = (TS_{dif}, [Params_i])$.

O restante do procedimento é idêntico aos passos 4 à 7 descritos para a concatenação de uma entrada não compactada.

5.3. Verificação do log de auditoria

A verificação do log de auditoria pode ser executada tanto pelo verificador V, quanto pelo servidor T. Para iniciar o procedimento, V requisita e obtém, de forma segura, de T a chave inicial de verificador. Com a chave em posse, V consegue calcular todas as chaves iniciais de todas as categorias de log através da função *SbC-EC.Ckg*. V requisita os arquivos de log de seu interesse para U e recalcula a tag de verificador $\mu_{A_{i,m}}$, pela Equação 1, de cada arquivo de log m. V então verifica se as tags são idênticas às que U armazena. Se sim, o log de auditoria está íntegro, caso contrário, isso implica que ao menos um componente de toda a sequência não está de acordo e, portanto, não está íntegro.

Como dito anteriormente, a estratégia de separação por categorias oferece uma vantagem de que, se V estiver interessado em apenas um tipo de log (p. ex., atividades de rede), ele só precisará requisitar e verificar o arquivo de log da categoria desejada, sem a necessidade de verificar todas as entradas de todos os arquivos de log. Na verificação feita por T, é necessário requisitar e verificar todos os arquivos de log. T realiza o mesmo procedimento de recalcular as tags de verificador para cada arquivo de log e, por fim, recalcula a tag da parte confiável pela Equação 2 e compara essa tag com a tag da parte confiável informada por U. Se as duas tags forem iguais, os arquivos de log estão íntegros, caso contrário, não estão.

6. Discussão de Segurança

O esquema SbC-EC MAC utiliza o esquema *FssAgg* como camada de autenticação para criar as tags de verificador. A cada nova entrada de log gerada, o protocolo recria a tag única, e de tamanho constante, para o arquivo de log correspondente. A segurança

do esquema *FssAgg* é provada nos trabalhos de Ma e Tsudik [Ma e Tsudik 2007] [Ma e Tsudik 2008] [Ma e Tsudik 2009b] [Ma e Tsudik 2009a]. Além disso, o esquema SbC-EC MAC é resistente ao ataque de *truncation* devido à propriedade "all-or-nothing" de verificação do arquivo de log herdado da autenticação *FssAgg*.

Por possuir a chave secreta inicial A_0, um usuário/verificador, V, autorizado e mal-intencionado é capaz de realizar alterações nas entradas de log e recriar as assinaturas agregadas (i.e., as tags de verificador) com o objetivo de esconder uma manipulação feita. Outros usuários autorizados, portanto, não poderão detectar essas alterações feitas por V e verificariam com sucesso os arquivos de log. A detecção da manipulação por V é possível apenas para o servidor T com ajuda da tag da parte confiável μ_j. Dessa forma, o esquema SbC-EC MAC é suscetível ao tipo ataque denominado *delayed detection* (detecção atrasada), devido ao fato de que nenhum V pode detectar prontamente uma manipulação realizada por outro V, e que a comunicação entre U e T só acontece em períodos de tempo indeterminados.

Devido a isto, a tag da parte confiável μ_j torna-se necessária para a segurança dos arquivos de log contra o atacante com privilégios, como V. Essa tag precisa ser atualizada após cada concatenação de uma nova entrada de log, possibilitando que qualquer alteração realizada nas tags de verificador seja detectada pelo servidor de coleta. μ_j é calculada a partir de uma função $MAC(X)$ após a geração de uma nova entrada de log, onde X é a concatenação de todas as tags de verificação em um determinado tempo t. μ_j é não-falsificável para os períodos antes de t, uma vez que a chave da parte confiável, B, evolui através de uma *pseudo-random function* após cada atualização dessa tag.

Por fim, salientamos que o esquema SbC-EC MAC oferece a propriedade de *forward-secrecy* e *forward secure stream integrity*. A propriedade *forward-secrecy* provê a permanência da confidencialidade das entradas de log concatenadas ao arquivo antes de uma intrusão, ou seja, mesmo que um atacante consiga invadir o sistema no tempo t, a confidencialidade das entradas geradas antes de t é mantida. Da mesma forma, a propriedade *forward secure stream integrity* também garante a integridade da sequencia das entradas do arquivo de log antes do momento t. Essas duas propriedades tem como base o mecanismo de evolução de chaves simétricas.

7. Avaliação

Nesta seção, o esquema SbC-EC MAC é avaliado de acordo com o ganho de armazenamento em memória em relação a outros esquemas de log seguro de criptografia simétrica existentes na literatura: (i) Schneier-Kelsey e (ii) FssAgg MAC. A seguir, descrevemos algumas premissas sobre o armazenamento necessário de cada item presente nos esquemas e, por fim, uma simulação é realizada com fins comparativos.

Seja M_{key} o espaço de armazenamento necessário para guardar as chaves privadas, M_{mac} o espaço necessário para guardar uma saída obtida de uma função $MAC()$ (p. ex.,

Tabela 1. Sobrecarga da autenticação dos esquemas de log seguro

	Sobrecarga	Ordem
Schneier-Kelsey	$M_{key} + N(M_{hash} + M_{mac})$	$O(N)$
FssAgg MAC	$2M_{key} + 2M_{hash}$	$O(1)$
SbC-EC MAC	$(c+1)M_{key} + cM_{hash} + M_{mac}$	$O(c)$

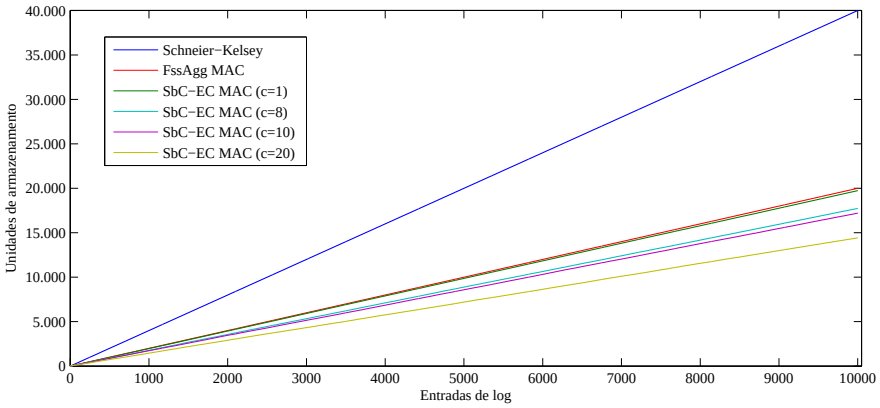

Figura 3. Simulação dos esquemas de log seguro em relação às unidades de armazenamento para N = 10.000.

assinaturas, tags) e M_{hash} o espaço necessário para guardar uma saída obtida de uma função hash $\mathcal{H}()$, dado N o número total de entradas de log geradas e c o número de categorias de log, verificamos na Tabela 1 a sobrecarga de armazenamento de autenticação associada a cada esquema. Pela tabela conclui-se que, a sobrecarga de autenticação do esquema de Schneier-Kelsey está diretamente ligada ao número de entradas logadas, uma vez que esse esquema mantém uma tag de autenticação para cada entrada. O esquema FssAgg MAC é o que implica na menor sobrecarga, utilizando apenas duas chaves e duas tags para autenticação de todo o arquivo de log. O esquema SbC-EC MAC apresenta uma sobrecarga de autenticação diretamente relacionada ao número de categorias de log estipuladas, pois, para cada categoria, uma chave e uma tag são mantidas.

Para avaliar o ganho de espaço de armazenamento do esquema SbC-EC MAC, realizamos uma simulação de geração de entradas de log a fim de calcular o tamanho total ocupado pelo log de auditoria protegido pelos esquemas testados. Para essa simulação, definimos um dispositivo capaz de registrar 400 mensagens de log diferentes (i.e. 400 IDs) e que segue uma distribuição normal para a geração de logs. Assumimos que, M_{key}, M_{mac} e M_{hash} são equivalentes a 1 unidade de armazenamento, o tamanho médio de uma entrada de log é de 2 unidades de armazenamento e o tamanho médio de uma entrada de log compactada é de 1 unidade de armazenamento. Os demais itens que compõe os logs de auditoria de cada esquema puderam ser desconsiderados, por não impactar significativamente no total de armazenamento ocupado. O gráfico da Figura 3 mostra o resultado de uma rodada da simulação de N = 10.000 entradas de log para cada esquema de log seguro em relação às unidades de armazenamento necessárias para guardar os arquivos de log.

Com o intuito de obter o tamanho médio e o ganho de armazenamento dos esquemas FssAgg MAC e SbC-EC MAC em relação ao esquema de Schneier-Kelsey, realizamos 1.000 rodadas de simulação para N = 10.000 entradas de log. Os resultados são mostrados na Tabela 2. Para os esquema de Schneier-Kelsey e FssAgg MAC, o armazenamento se mantém constante para todas as rodadas, uma vez que eles detém o mesmo comportamento independente das entradas de log geradas. O FssAgg MAC oferece um ganho de armazenamento de 50% em relação ao esquema anterior. Para o esquema SbC-EC MAC, observa-se que, quanto maior o número de categorias c estipulado, menor são as unidades de armazenamento necessárias para guardar o log de auditoria. Isso é expli-

Tabela 2. Resultado da simulação em 1.000 rodadas

	Tamanho médio (un. de armazenamento)	Desvio padrão (σ)	Ganho de arm. (%)
Schneier-Kelsey	40.000	0	-
FssAgg MAC	20.000	0	50,00
SbC-EC MAC (c =1)	19.719	16,60	50,70
SbC-EC MAC (c =8)	17.743	42,56	55,64
SbC-EC MAC (c =10)	17.181	46,42	57,05
SbC-EC MAC (c =20)	14.379	60,39	64,05

cado pelo fato de que há mais probabilidade para $ID_i = ID_{i-1}$ quando há mais divisões, agrupando entradas de log cada vez mais semelhantes. Dessa forma, nosso esquema permite um maior ganho de armazenamento em relação ao FssAgg MAC. Essa avaliação evidencia a importância da categorização de mensagens de log no sistema para o esquema SbC-EC MAC e que, quanto maior a granularidade dessa categorização, maior o ganho de armazenamento.

8. Conclusão e Trabalhos Futuros

Neste trabalho, apresentamos uma proposta de um novo esquema de estruturação de log seguro, o SbC-EC, apropriado para dispositivos com restrições de memória de armazenamento e de comunicação em rede. O esquema de estruturação SbC-EC é fundamentado em duas características com o objetivo de reduzir o espaço necessário para armazenar entradas de log: (i) Separação por Categoria (*Split by Category*, SbC) e (ii) Compactação de Entradas de log (*Entry Compaction*, EC).

Apresentamos, também, um novo esquema de log seguro baseado nessa nova estruturação, o SbC-EC MAC. O esquema SbC-EC MAC utiliza as primitivas da criptografia simétrica e o esquema de autenticação *FssAgg*, criado por Ma e Tsudik [Ma e Tsudik 2007], para criar as tags de autenticação do log de auditoria. Descrevemos as funções presentes no SbC-EC MAC, como é realizada a inicialização e verificação dos arquivos de log, a concatenação e a compactação das entradas de log. Além disso, são feitas algumas considerações sobre a segurança do esquema, onde expomos algumas características como a *forward-secrecy*, a *forward secure stream integrity* e a suscetibilidade ao ataque de *delayed detection*, causado por um usuário privilegiado. Por fim, demonstramos uma avaliação do esquema SbC-EC MAC em relação ao ganho de armazenamento, comparando-o a outros dois esquemas simétricos existentes na literatura.

Como trabalhos futuros, consideramos adicionar características ao esquema, levando em conta a criticidade da informação do log e desempenho computacional necessário para proteção dos arquivos de log. Assim como elaborar, baseado na estruturação SbC-EC, um esquema de log seguro para o domínio da criptografia de chave-pública.

Referências

Accorsi, R. (2011). BBox: A distributed secure log architecture. In *proceedings of the 7th European Workshop on Public Key Infrastructures, Services and Applications*, páginas 109–124.

Accorsi, R. (2013). A secure log architecture to support remote auditing. *Mathematical and Computer Modelling*, 57(7-8):1578–1591.

Bellare, M., Canetti, R., e Krawczyk, H. (1996). Keying hash functions for message authentication. páginas 1–15. Springer-Verlag.

Bellare, M. e Yee, B. (1997). Forward integrity for secure audit logs. Technical report, Computer Science and Engineering Department, University of California at San Diego.

Bellare, M. e Yee, B. (2003). Forward-security in private-key cryptography. *Topics in Cryptology-CT-RSA 2003*, páginas 1–24.

Holt, J. E. (2006). Logcrypt: Forward Security and Public Verification for Secure Audit Logs. In *Proceedings of the 2006 Australasian Workshops on Grid Computing and e-Research - Volume 54*, ACSW Frontiers '06, páginas 203–211, Darlinghurst, Australia, Australia. Australian Computer Society, Inc.

International Electrotechnical Commission (2011). ISA - Security for Industrial Automation and Control Systems - Technical Security Requirements for IACS Components - Part 4.

Ma, D. e Tsudik, G. (2007). Extended Abstract: Forward-Secure Sequential Aggregate Authentication. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, SP '07, páginas 86–91, Washington, DC, USA. IEEE Computer Society.

Ma, D. e Tsudik, G. (2008). A New Approach to Secure Logging. In *Proceeedings of the 22Nd Annual IFIP WG 11.3 Working Conference on Data and Applications Security*, páginas 48–63, Berlin, Heidelberg. Springer-Verlag.

Ma, D. e Tsudik, G. (2009a). A New Approach to Secure Logging.

Ma, D. I. e Tsudik, G. (2009b). A New Approach to Secure Logging. *ACM Transactions on Storage*, 5(1):2:1—-2:21.

National Institute of Standards and Technology (2001). Announcing the Advanced Encryption Standard (AES).

National Institute of Standards and Technology (2012). FIPS PUB 180-4, Secure Hash Standard, Federal Information Processing Standard (FIPS), Publication 180-4. Technical report, Department Of Commerce.

Oprea, A. e Bowers, K. D. (2009). Authentic time-stamps for archival storage. In *Proceedings of the 14th European Conference on Research in Computer Security*, ESORICS'09, páginas 136–151, Berlin, Heidelberg. Springer-Verlag.

Schneier, B. e Kelsey, J. (1998). Cryptographic Support for Secure Logs on Untrusted Machines. In *Proceedings of the 7th Conference on USENIX Security Symposium - Volume 7*, SSYM'98, página 4, Berkeley, CA, USA. USENIX Association.

Schneier, B. e Kelsey, J. (1999). Secure audit logs to support computer forensics. *ACM Transactions on Information and System Security*, 2(2):159–176.

Yavuz, A. A. e Ning, P. (2009). BAF: An Efficient Publicly Verifiable Secure Audit Logging Scheme for Distributed Systems. In *Computer Security Applications Conference, 2009. ACSAC '09. Annual*, number ii, páginas 219–228.

Yavuz, A. a., Ning, P., e Reiter, M. K. (2012). BAF and FI-BAF: Efficient and Publicly Verifiable Cryptographic Schemes for Secure Logging in Resource-Constrained Systems. *ACM Transactions on Information and System Security*, 15(2):1–28.

