Anonimização de Traços Wi-Fi com Controle do K-Anonimato em Ambientes de Execução Confiável

Pedro V. Rubinstein, Fernando Dias de M. Silva, Guilherme A. Thomaz, Miguel Elias M. Campista e Luís Henrique M. K. Costa

[1]Grupo de Teleinformática e Automação (GTA)
Universidade Federal do Rio de Janeiro(UFRJ)
Caixa Postal 68.504 – 21.941-972 – Rio de Janeiro – RJ – Brazil

{prubinstein,fernandodias,guiaraujo,miguel,luish}@gta.ufrj.br

Resumo. *O monitoramento Wi-Fi possui diversas aplicações, como localização e contagem de pessoas. Porém, é fundamental preservar a privacidade dos usuários. Por outro lado, dispositivos de coleta que operam em locais públicos e sem supervisão estão sujeitos a ataques que podem comprometê-los. Este trabalho apresenta um sniffer que utiliza ambientes de execução confiáveis no processo de coleta de traços para anonimizar os dados no dispositivo de forma segura. Assim, o anonimato de dados previamente armazenados é preservado, mesmo na presença de atacantes que controlem o sistema operacional do dispositivo. Para isso, o sniffer implementado garante um nível de k-anonimato dos endereços MAC. Ademais, a implementação realizada otimiza o uso de memória RAM para apenas 10 MB, considerados os ambientes tradicional e de execução confiável, possibilitando a implementação em dispositivos limitados. Os experimentos revelam que a latência adicional das rotinas criptográficas no ambiente confiável não impede o monitoramento Wi-Fi em cenários realistas.*

Abstract. *Wi-Fi monitoring leverages several applications, such as localization and counting people. Yet, preserving user privacy remains a critical concern. At the same time, data collection devices operating in public and unsupervised environments are vulnerable to attacks which may compromise them. This work presents a Wi-Fi sniffer that leverages trusted execution environments during the sniffing process to securely anonymize data on the device. As a result, the anonymity of previously stored data is preserved even in the presence of attackers with full control over the device's operating system. To this end, the proposed sniffer enforces a k-anonymity level for MAC addresses. Furthermore, the implementation optimizes memory usage to only 10 MB of RAM considered the traditional and trusted execution environments, enabling deployment on resource constrained devices. Experimental results show that the latency added by cryptographic routines within the trusted environment does not prevent Wi-Fi monitoring in realistic scenarios.*

1. Introdução

A coleta passiva de quadros do padrão IEEE 802.11 constitui uma alternativa de baixo custo para a obtenção de dados utilizados em aplicações como predição de mobilidade. Entretanto, regulamentações como a Lei Geral de Proteção de Dados (LGPD) proíbem a persistência de dados pessoais sensíveis sem o consentimento do titular. Assim, são necessários mecanismos de anonimização capazes de substituir identidades reais

por pseudônimos, reduzindo os riscos associados à exposição dos dados armazenados. Métodos tradicionais, como a aplicação de uma função *hash* e a truncagem de identificadores, fazem com que ao menos k identidades distintas sejam mapeadas em um único pseudônimo, fornecendo um nível de proteção conhecido por k-anonimato. Tipicamente, este processo é aplicado logo após a coleta dos dados (*online*), garantindo que apenas pseudônimos sejam armazenados em disco. Como os quadros provenientes de diferentes dispositivos chegam de maneira aleatória, não há como escolher o valor ideal de k *a priori*, tornando o nível de privacidade do conjunto de dados final imprevisível: o nível de anonimato em um conjunto depende do número final de identificadores distintos coletados, desconhecido durante a coleta. A alternativa *offline* resolve esse problema ao armazenar os dados criptografados em disco e aplicar a anonimização *a posteriori*, onde é possível ter uma visão global dos traços para que a anonimização alcance um valor de k pré-definido. Esta abordagem, no entanto, possui uma limitação crítica ainda pouco explorada na literatura: um atacante com acesso indiscriminado a todo o Sistema Operacional (SO) pode obter as chaves criptográficas utilizadas para armazenamento do traço coletado antes da anonimização, recuperando todo o histórico de dados coletado até então.

Este trabalho propõe uma arquitetura de coleta passiva de quadros de gerenciamento (*management frames*) do padrão IEEE 802.11 e anonimização de campos que se diferencia de outros trabalhos ao garantir a segurança mesmo na presença de um atacante com nível de acesso privilegiado ao SO. A principal inovação deste trabalho é o uso de Ambientes de Execução Confiáveis (*Trusted Execution Environment* – TEEs) para que os campos sensíveis sejam temporariamente armazenados com uma chave acessível apenas dentro de uma região isolada na memória do SoC (*System-on-Chip*). Assim, é proposto um algoritmo de geração do conjunto de dados anonimizado dentro do TEE, dividido em etapas *online* e *offline*, que permite o controle exato do parâmetro k para definir o nível de proteção por k-anonimato.

A implementação usa um Raspberry Pi 3, dispositivo com recursos limitados e compatível com o TEE ARM TrustZone e o sistema operacional de tempo-real (*Real Time Operating System* – RTOS) seguro, o OP-TEE. O *sniffer* coleta quadros do sub-tipo *probe request*, utilizados para busca de pontos de acesso por estações, que contém campos privados, como endereços de enlace. Ademais, foi adotado um conjunto de otimizações que viabilizam a execução da proposta em dispositivos com recursos computacionais limitados. Essas otimizações incluem modificações no algoritmo de anonimização e o uso eficiente de APIs da linguagem Rust, possibilitando o funcionamento com apenas 10 MB de memória RAM em um cenário real de coleta em um shopping movimentado. Os resultados revelam que, no pior caso, o tempo do *sniffer* em TEE é até 18 vezes superior ao tempo de uma implementação sem TEE. Adotando uma estratégia de anonimização em lotes, esse fator foi reduzido para 10 vezes. Ademais, o uso do TEE não aumenta significativamente o percentual de quadros perdidos em cenários de alta taxa. Assim, a proposta aumenta significativamente a privacidade sem inviabilizar cenários de coleta em alta taxa.

Este trabalho está assim organizado. A Seção 2 descreve os modelos de ataque considerados, mecanismos de anonimização e TEEs. A Seção 3 discute trabalhos relacionados. A Seção 4 descreve a arquitetura do sniffer proposto, enquanto a Seção 5 destaca os desafios práticos de *hardware* e *software*. A Seção 6 descreve a metodologia de avaliação e resultados obtidos. A Seção 7 conclui o artigo e apresenta trabalhos futuros.

2. Fundamentação Teórica

Esta seção apresenta os principais ataques aos sistemas de coleta de endereços de rede, bem como as estratégias de anonimização tradicionais e os conceitos necessários para a operação em ambientes TEE.

2.1. Anonimização de Endereços de Rede

Mesmo sem o conhecimento da identidade do titular, os dispositivos ainda podem ser identificados através de seu endereço MAC e da lista de SSIDs (*Service Set Identifier* – SSIDs) que consultam através de quadros de gerenciamento Wi-Fi específicos.

Para um conjunto de endereços possíveis $\mathcal{M}$ e pseudônimos $\mathcal{P}$, a função $f : \mathcal{M} \rightarrow \mathcal{P}$ mapeia a identidade real do usuário a um pseudônimo. Chamamos essa função de função de anonimização. Quando $|\mathcal{M}| = |\mathcal{P}|$, é possível fazer a associação 1 para 1 de endereços, o que é considerado uma fraqueza. Isso porque se o atacante descobre o endereço de um usuário referente a um pseudônimo, ele consegue descobrir o endereço não anonimizado de todas as ocorrências anteriores e futuras daquele mesmo dispositivo. Isso impede, por exemplo, que criptografia simétrica com chaves efêmeras e determinísticas seja uma solução viável, pois, por mais que não se saiba associar um endereço a um determinado pseudônimo sem o conhecimento da chave, a abordagem mantém o mapeamento 1 para 1. Então, assumindo a ausência de colisões, todas as ocorrências de algum pseudônimo no conjunto de dados são resultantes de um, e apenas um, endereço.

O modelo teórico de anonimização mais adotado na literatura é o k-anonimato, no qual qualquer pseudônimo p_i é associado a no mínimo k endereços, tornando-os indistinguíveis entre si. Para isso, é condição necessária que $|\mathcal{M}| \gg |\mathcal{P}|$. Assim, mesmo que o atacante consiga fazer uma associação em seu subconjunto, ele não obterá novas informações do conjunto de dados original, já que ele pode até saber quais são os k endereços que mapeiam em um mesmo pseudônimo, mas, ao observar o pseudônimo, ele não saberá distinguir qual dos k endereços geraram aquela entrada.

A técnica mais tradicional de anonimização para garantia de k-anonimato é conhecido como *hash-and-truncate*, que consiste no uso de uma função de *hash*, no truncamento do resultado para um número pequeno de bits b para induzir colisões e, por fim, no descarte do quadro original [Silva et al. 2022]. Esta técnica pode ser implementada usando um método *online*, no qual um endereço em claro é anonimizado e descartado assim que chega, ou *offline*, que mantém os endereços brutos armazenados e aplica a anonimização somente ao final do processo de captura.

Em termos de privacidade, no método *online*, não se sabe quais ou quantos endereços serão capturados *a priori*, fazendo com que a escolha do hiperparâmetro b seja feita cegamente. Assim, o valor de k obtido ao final do processo é uma variável aleatória. No método *offline*, o problema de escolha de k inexiste dado que o conjunto de dados completo é conhecido, permitindo escolher b de forma a garantir o nível de k-anonimato.

Em termos de desempenho, a elevada taxa de quadros por segundo na interface sem-fio torna impraticável a manutenção de todos os quadros na RAM. Assim, para permitir a anonimização *offline* em dispositivos com memória restrita, os quadros não anonimizados devem ser escritos em disco (*dump*). Entretanto, esta abordagem é vulnerável a um atacante com acesso ao disco em tempo real, conforme abordado a seguir.

2.2. Ataques à Coleta de Endereços de Rede

Assume-se um cenário no qual um *sniffer* é posicionado em um ambiente público e sem supervisão. Neste caso, os principais tipos de ataque são:

- Ataques à disponibilidade: encerramento prematuro do processo de coleta, ou mesmo apagamento dos dados coletados até então.
- Ataques à integridade: modificação do histórico de dados coletados.
- Ataques à confidencialidade: criação de uma cópia não anonimizada do conteúdo observado pelo *sniffer*.
- Ataques de reidentificação: associação entre entradas do conjunto de dados anonimizado com o não anonimizado, efetuada após um ataque à confidencialidade.

O primeiro caso é uma negação de serviço, fora do escopo desse trabalho. O trabalho foca em evitar o comprometimento da confidencialidade e integridade dos endereços dos usuários, evitando assim ataques de reidentificação. Assume-se que todos os ataques possam ocorrer em dois níveis de força:

- **Nível 1** - O atacante só possui acesso a dados coletados após o ataque, por exemplo posicionando um segundo *sniffer* no mesmo ambiente do *sniffer* legítimo (*spoofing*), uma alternativa de ataque de difícil mitigação.
- **Nível 2** - O atacante invade o *sniffer* legítimo e possui acesso ao histórico completo de dados coletados. Há duas mitigações possíveis. A primeira, o *sniffer* alvo pode anonimizar os endereços recebidos *online*, não mantendo histórico das identidades. A outra opção é encriptar os traços com uma chave configurada pelo dono do *sniffer*.

Este trabalho foca em melhorias na proteção contra ataques de Nível 2. Proteção completa contra *spoofing* (ataques de Nível 1) costuma ser impraticável, uma vez que o próprio conjunto de padrões IEEE 802.11 pressupõe modulação em faixas de frequências não-licenciadas, bem como mecanismos de controle de acesso baseados em quadros de gerenciamento não encriptados (portanto públicos), como os *probe request* difundidos em *broadcast* para todos os dispositivos na região de alcance.

Enquanto o método *online* não garante um nível de k-anonimato determinístico, o método *offline* exige a escrita dos quadros não anonimizados em disco devido à limitações de memória RAM. Nesse caso, um atacante pode realizar ataques simples de leitura e escrita em disco. A solução típica é a encriptação com uma chave. Nesse caso, porém, há a premissa de que o atacante não possui acesso privilegiado ao SO, impossibilitando a leitura da chave. Este trabalho considera um modelo de atacante mais poderoso, com acesso completo ao SO do *sniffer*. Assim, o atacante não só acessa qualquer arquivo, como também lê os conteúdos em memória, como chaves criptográficas.

2.3. Ambientes de Execução Confiável

A Computação Confidencial (*Confidential Computing*) consiste no uso de recursos de *hardware* para prover segurança. Entre as soluções de computação confidencial, destacam-se os Ambientes de Execução Confiáveis (TEEs), nos quais dados são processados em um ambiente isolado, denominado enclave. Diferente de soluções tradicionais de criptografia, o uso de TEEs mantém a confidencialidade e a integridade dos dados localizados no TEE mesmo na presença de atacantes que controlam o SO do hospedeiro.

A solução de TEE utilizada em *Systems-on-Chip* (SoCs) ARM é o TrustZone-A, referido daqui em diante como TrustZone ou TZ [Ngabonziza et al. 2016]. Em um

computador com TZ, o *software* e *hardware* estão divididos em dois mundos: inseguro e seguro [Göttel et al. 2019]. O mundo inseguro, ou ambiente de execução "rico" (*Rich Execution Environment* – REE) é o ambiente de execução tradicional de um SO, que executa aplicações de uso geral. Já o mundo seguro (TEE) executa um SO enxuto, como o OP-TEE, que é um sistema operacional de tempo-real (RTOS). Esses dois sistemas executam no mesmo nível de privilégio, mas as faixas de endereços de memória e dispositivos de entrada e saída (*Input/Output* – I/O) são separadas, com bloqueio de acesso indevido por *hardware*. O modo do processador (seguro e inseguro) é identificado pelo bit mais significativo do registrador de configurações de segurança da CPU. A transição entre os mundos é feita por um *software* com mais alto nível de privilégio, chamado monitor.

Enquanto o Linux de uso geral implementa *drivers* para acesso a dispositivos de entrada e saída (I/O) por meio de chamadas de sistema, o OP-TEE apenas inicia aplicações confiáveis (*Trusted Applications* – TAs) mediante pedido de uma aplicação do REE quando for necessário executar processamentos críticos para segurança. Esses processamentos incluem encriptação de dados sensíveis, o processamento de dados em claro e a encriptação dos resultados.

3. Trabalhos Relacionados

Diversos trabalhos fazem uso de *sniffers* para a implementação de aplicações de monitoramento [Capponi et al. 2019]. Algumas das aplicações existentes envolvem contagem de multidões em ambientes internos [Manjappa et al. 2019], contagem de passageiros em sistemas de transporte públicos [Wang et al. 2021, Reichl et al. 2018], estimação de interesse em exposições [Das et al. 2022], contagem de pessoas [Sakib et al. 2014, Verma e Singh 2019] e posicionamento em ambientes internos [Ribeiro et al. 2021]. O monitoramento do Wi-Fi possui diversas aplicações práticas que, para serem aproveitadas em cenários reais, dependem de técnicas de preservação da privacidade para que possam ser aplicadas, dada a existência de legislações de proteção de dados.

Para lidar com a privacidade, alguns trabalhos como [Tan e Kotz 2010] implementam um sistema de monitoramento que realiza encriptação e envio para um servidor central. Essa abordagem utiliza o ingresso em uma rede Wi-Fi como consentimento para a coleta de dados e, assim, não precisa destruir os dados originais. Porém, esse método utiliza pontos de acesso e faz a transmissão contínua dos pacotes coletados, o que prejudica a banda disponível. O método de [Determe et al. 2022] faz uso de um servidor central para gerar valores temporários em intervalos regulares para tornar a anonimização dinâmica e os identificadores limitados a intervalos de tempo curtos. Isso, porém, não impede que atacantes obtenham controle do dispositivo e possam assim obter tanto o valor temporário armazenado no servidor quanto o armazenado no dispositivo. O trabalho em [Silva et al. 2022] mostra o desempenho de um *sniffer* que implementa a técnica de *hash* e truncamento para anonimização de dispositivos de forma online. Esse método oferece um k-anonimato para valores de truncamento baixos, mas o trabalho não apresenta uma relação direta entre esses valores.

A proteção de endereços é necessária até mesmo para dispositivos que atribuem endereços MAC aleatórios à interface Wi-Fi: o trabalho de [Tan e Gary Chan 2021] mostra como é possível recuperar a informação de trajetória de dados supostamente anonimizados. Assim, é necessário anonimizar todos os endereços sob um mesmo algoritmo para garantir níveis mínimos e justos de privacidade. O trabalho de [Martin et al. 2017]

critica métodos de anonimização que fazem uma associação direta entre pseudônimos e endereços, já que alguns algoritmos utilizados (como o *hash* e truncamento) podem ser calculados para todo o espaço de endereços MAC, por exemplo do padrão EUI-48 usado pelos protocolos de redes locais da família IEEE 802.

Há poucos trabalhos na literatura que se preocupam em atender de forma simultânea três requisitos, normalmente conflitantes: controle exato do nível de k-anonimato, baixo uso de memória e proteção contra atacantes que controlam o SO completo. Diferente do estado da arte, este trabalho propõe uma arquitetura que se serve de TEEs para proteger a anonimização de endereços na camada 2 em *sniffers* Wi-Fi, com baixo uso de memória. A contribuição do artigo está na proteção contra atacantes com controle total do SO do *sniffer* que buscam obter identidades dos quadros coletados até então. Os resultados revelam a sobrecarga introduzida pelo TEE devido às operações de anonimização em termos de tempo e taxa de perdas de pacotes, não avaliadas na literatura.

4. Arquitetura para Anonimização Confiável de Endereços de Rede

A anonimização proposta consiste em duas etapas: a primeira etapa faz a coleta e a contagem *online* de endereços e a segunda etapa atribui pseudônimos a endereços de forma *offline*. Entre esses processos, o arquivo de captura é salvo com um endereço criptografado único para cada entrada. Um diagrama de visualização da arquitetura pode ser encontrado na figura 1, sendo detalhada ao longo das seções 4 e 5.

Durante a etapa *online*, é necessário manter um registro de todos os endereços brutos observados dentro do período de coleta. Esse registro fica em uma tabela de contagem (ex. Tabela 1), que associa endereços reais m_i a nomes temporários $t_j^{[m_i]}$. Cada valor de j representa um novo nome temporário associado ao endereço m_i, definido na j-ésima vez que o mesmo endereço aparece no processo de coleta. Assim, assumindo uma função criptográfica $E(c, v)$ que encripta v com a chave c, o cálculo dos nomes temporários é representado por

$$t_j^{[m_i]} = E(s_i, t_{j-1}^{[m_i]}),$$

onde s_i é uma semente para cada m_i, e t_{j-1} é o último nome aleatório calculado. O primeiro valor pode ser calculado como $t_0^{[m_i]} = E(s_i, m_i)$.

A tabela de contagem é mantida na memória RAM do TEE, garantindo que os endereços reais estejam protegidos de atacantes com alto nível de privilégio. Na prática, essa tabela pode ser estendida para armazenar informações como o SSID, com sementes separadas. Deve-se utilizar um espaço de nomes temporários grande para evitar colisões ao longo da contagem de endereços. Ademais, as sementes s_i são temporárias e produzidas por um gerador de números pseudo-aleatórios criptograficamente seguro (*Cryptographically Secure Pseudo-Random Number Generator* – CSPRNG).

Ainda na etapa *online*, é montada uma segunda tabela, de associação (ex. Tabela 2) usada para reconstrução do arquivo final. Ela mapeia nomes temporários em pseudônimos, assegurando que todos os nomes temporários associados a um mesmo endereço recebam o mesmo pseudônimo. Além disso, a tabela garante que cada pseudônimo seja compartilhado por, no mínimo, k endereços distintos. A imposição do valor de k só é possível graças ao fato de que a anonimização só é realizada após obter uma visão global dos diferentes endereços coletados (tabela de contagem).

Os TEEs encontrados em microcontroladores e SoCs utilizados em IoT costu-

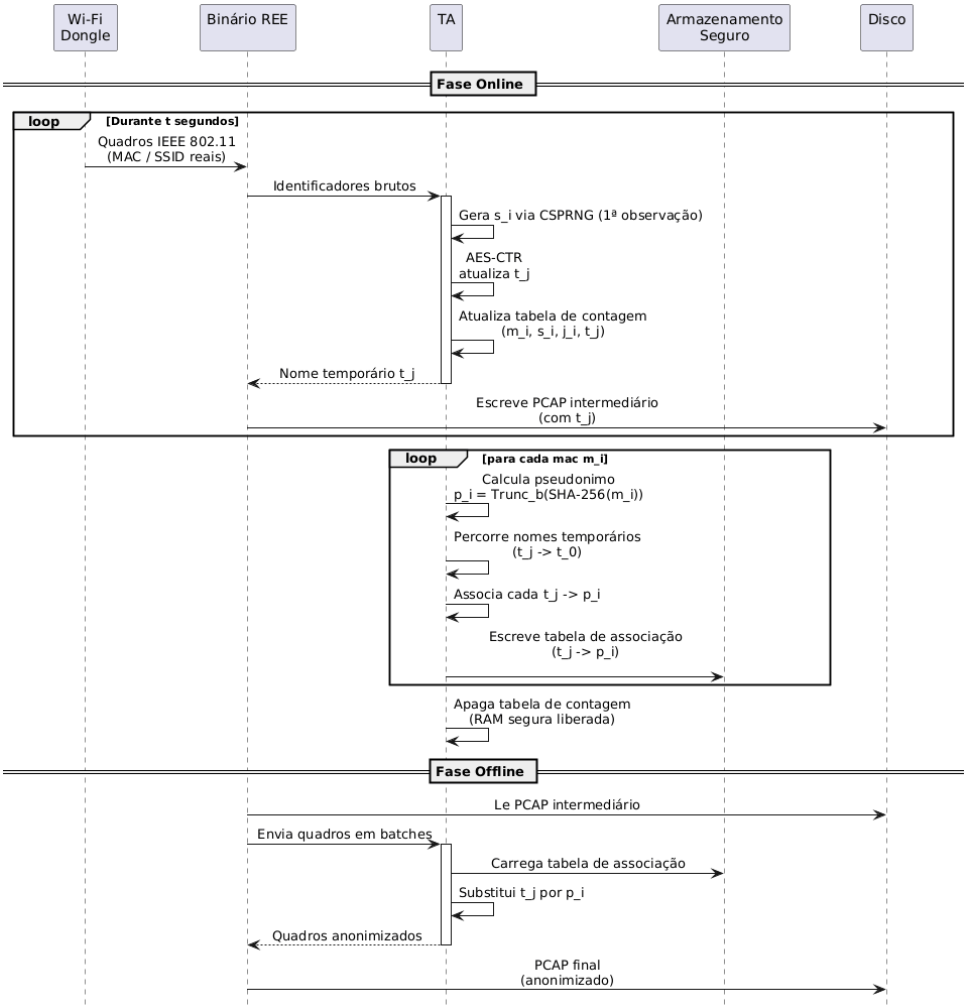

Figura 1. Diagrama representando a implementação da arquitetura proposta.

mam possuir limitações severas de memória. Assim, armazenar todas as tabelas em RAM pode limitar a duração máxima da captura para evitar estouro do *heap*. Assim, na implementação realizada cada entrada da tabela de associação é escrita em disco à medida que a tabela é montada. Para garantir segurança, os dados são encriptados com uma chave, chamada de chave de selagem, gerada pelo CSPRNG e acessível apenas para o TEE. A fase *online* termina com o apagamento da tabela de contagem em memória, liberando espaço para a etapa *offline*.

Na segunda etapa (*offline*), a tabela de associação é armazenada em memória. Devido ao seu tamanho, pode ser necessário realizar o carregamento parcial da estrutura, com a substituição dinâmica de seus segmentos em memória. O *sniffer* então envia os pacotes armazenados no arquivo de captura temporário para o TA, que substitui os nomes temporários pelos pseudônimos correspondentes, como definido na tabela de associação.

5. Implementação da Arquitetura Proposta

Esta seção descreve como a arquitetura de anonimização confiável, proposta na seção anterior, foi implementada em uma plataforma *single-board computer* comercial.

Tabela 1. Exemplo de tabela de contagem utilizada no sniffer.

Endereço real (m)	Semente (s)	Nº observações (j)	Último nome temporário (t_j)
FF:DE:21:22:33:44	al3ryamg	45	ls23lkadsfnl
FF:DE:21:22:33:65	a0ç8tlrt	45	lsdfgjdgzsrg
FF:DE:21:22:33:86	al87urh2	45	asçln3rassnl

Tabela 2. Exemplo de tabela de associação utilizada no sniffer.

Nome temporário (t_j)	Pseudônimo
ls23lkadsfnl	p_1
lsdfgjdgzsrg	p_2
asçln3rassnl	p_1

5.1. Hardware e Software

O *hardware* utilizado é um Raspberry Pi 3B, compatível com o TEE ARM TZ. A captura de tráfego é realizada utilizando um adaptador (*dongle*) Wi-Fi USB TP-Link TL-WN722N v1.0, escolhido por sua facilidade de configuração em modo monitor.

O SO instalado é o OP-TEE versão 4.8.0. A distribuição oficial inclui um SO para o TEE (o OP-TEE propriamente dito) e o *Buildroot* no REE. Entretanto, essa configuração padrão é minimalista e não fornece suporte adequado a interfaces Wi-Fi externas nem a ferramentas de desenvolvimento avançadas. Para viabilizar a captura de tráfego IEEE 802.11 e facilitar a instrumentação experimental, substituiu-se o sistema de arquivos raiz (*rootfs*) do REE por uma distribuição mínima do Ubuntu Mantic, mantendo-se o OP-TEE no mundo seguro. Durante o processo de compilação do OP-TEE, o *kernel* do REE é configurado para incluir o pacote `linux-firmware`, permitindo o carregamento dos *drivers* necessários para o adaptador Wi-Fi.

A ferramenta desenvolvida implementa o modelo proposto de anonimização, suportando anonimização de diferentes identificadores. Esses identificadores podem ser: endereço MAC do transmissor, endereço MAC do receptor, endereço MAC do ponto de acesso ou o SSID da rede. A arquitetura é composta por um binário executado no REE, responsável pela captura de pacotes IEEE 802.11, e por uma TA, responsável pela geração de nomes temporários, anonimização e armazenamento seguro. O código foi escrito na linguagem Rust utilizando o Teaclave TrustZone SDK [Project 2023]. A comunicação entre REE e TEE é realizada por meio das APIs do TEE expostas pelo Teaclave.

5.2. Implementação da Fase Online

Na fase *online*, o binário do sniffer no REE acessa a interface sem-fio pelo *driver*, extrai os campos relevantes dos quadros Wi-Fi e encaminha os identificadores à TA no TEE. Para cada endereço MAC ou SSID observado, a TA mantém na RAM a semente (s_i), o contador de observações e o último nome temporário gerado, conforme a Tabela 1. Com base nesse estado, a TA gera os nomes temporários, que são retornados ao REE e substituem os identificadores reais no arquivo PCAP intermediário. Esse arquivo preserva todas as demais informações de camada física e enlace. Ao final da coleta, a estrutura de dados que implementa a tabela é serializada e escrita no armazenamento seguro do OP-TEE, responsável por encriptar e decriptar os dados de forma transparente, permitindo o processamento posterior sem exposição dos identificadores reais. Em seguida, o TEE calcula os pseudônimos finais e mapeia nomes temporários em pseudônimos, conforme a Tabela 1.

O nível de truncagem é ajustado para garantir o parâmetro desejado de k-anonimato. Essa tabela é guardada no armazenamento seguro de forma persistente. Devido às restrições de memória do enclave, a construção da tabela é incremental, evitando carregar toda a estrutura em memória volátil de uma vez.

Para cada identificador real observado (endereço MAC ou SSID), a TA mantém uma entrada de estado $\langle m_i, s_i, j_i, t_j^{[m_i]} \rangle$, onde m_i é o identificador real, s_i é uma semente aleatória de 128 bits, j_i é o contador de observações e $t_j^{[m_i]}$ é o último nome temporário gerado. Na primeira observação de m_i, uma semente $s_i \leftarrow \{0,1\}^{128}$ é gerada por um gerador CSPRNG no TEE, e o primeiro nome temporário é calculado como:

$$t_0^{[m_i]} = \text{AES-CTR}(s_i, m_i, 0).$$

O AES-CTR (*Advanced Encryption Standard - Counter Mode*) refere-se ao modo de operação da cifra simétrica AES implementado na biblioteca Rust utilizada para programar a TA. O primeiro argumento é a chave, o segundo é o texto em claro (*plaintext*) e o terceiro é um contador, incrementado a cada novo quadro, começando em zero. Para quadros subsequentes, o nome temporário é atualizado recursivamente:

$$t_j^{[m_i]} = \text{AES-CTR}(s_i, t_j^{[m_i]}, j_i)$$

Dessa forma, cada observação gera um identificador pseudoaleatório distinto e criptograficamente desvinculado dos anteriores. Os nomes temporários de endereços MAC são truncados para 48 bits, preservando o formato estrutural de um endereço MAC EUI-48 válido. Essa truncagem não compromete a segurança, pois o valor cifrado completo nunca é exposto fora do enclave. Ressalta-se que a mesma abordagem pode ser estendida para endereços EUI-64 por meio de adaptações simples no código.

5.3. Implementação da fase offline

Durante a fase *offline*, o arquivo PCAP intermediário é reprocessado. O TEE carrega do armazenamento persistente seguro a tabela de associação entre nomes temporários e pseudônimos. O REE envia o arquivo ao TEE sequencialmente e, para cada entrada, o TEE substitui o nome temporário pelo pseudônimo correspondente, consultando a tabela de associação. Em seguida, o TEE retorna os dados anonimizados ao REE, que os escreve em um novo arquivo PCAP anonimizado. Esse processamento é realizado em lotes (*batches*), evitando o carregamento completo do arquivo em memória segura. Todas as strings que envolvem a leitura e escrita de arquivos compartilham um único *buffer* pré-alocado de modo a economizar espaço com *heap* na RAM. Também é importante notar que a etapa conta com um cache LRU (*Least Recently Used*) pré-alocado, de aproximadamente 6 MB, para evitar custosas leituras de arquivo no TEE.

Os pseudônimos finais são derivados a partir de uma função *hash* criptográfica SHA-256(m_i) aplicada ao identificador real m_i. O SHA-256 (*Secure Hash Algorithm* com blocos de 256 bits) é uma função *hash* popular, implementada na biblioteca Rust da TA. O pseudônimo é definido pelo tradicional método *hash*-truncagem:

$$p_i = \text{Trunc}_b\left(H(m_i)\right),$$

onde $\text{Trunc}_b(\cdot)$ representa a truncagem nos b bits mais significativos. Assuma que foram

observados n identificadores distintos e que o SHA-256 é uniforme. O número máximo de pseudônimos distintos é 2^b, e o valor esperado de identificadores por pseudônimo é:

$$\mathbb{E}[k] = \frac{n}{2^b}.$$

Para garantir $\mathbb{E}[k] \geq k_{\min}$, impõe-se:

$$b \leq \left\lfloor \log_2 \frac{n}{k_{\min}} \right\rfloor.$$

O valor de b é calculado dessa forma, ajustando o nível de anonimização ao tamanho real do conjunto de dados. Menores b induzem colisões controladas entre identificadores, aumentando o grau de indistinguibilidade entre usuários no conjunto de dados final.

5.4. Análise de Uso de Memória

A configuração padrão do OP-TEE para o Raspberry Pi 3B inclui 16 MB de memória dedicada ao TEE. Desse total, a configuração padrão do Teaclave aloca 32 kB para *heap* e 2 kB para a pilha (*stack*) de cada TA. Para garantir que os dados sensíveis permaneçam integralmente dentro da TA, o limite de *heap* foi expandido para 512 kB, valor máximo permitido pelo compilador. Esse espaço permite armazenar 2^{16} MACs EUI-48 únicos durante a fase *online*. Em um teste em cenário real de coleta realizado em um shopping com grande fluxo de pessoas, uma semana antes do natal de 2025, esse limite correspondeu a aproximadamente a uma hora de captura contínua. O consumo total de memória do sniffer implementado é de ≈3,4 MB no REE e ≈6,6 MB no TEE. Essa configuração representa um cenário restritivo de memória, o que motiva o uso de processamento em fluxo, a serialização incremental das estruturas internas em memória persistente segura e o compartilhamento de *buffers* e caches, de modo a evitar manter grandes estruturas em memória volátil.

6. Avaliação de Desempenho da Proposta

O incremento de segurança obtido com a utilização de cálculos no ambiente TEE impõe uma sobrecarga ao sniffer proposto. Esta seção investiga o impacto desta sobrecarga na implementação realizada. Primeiramente, é apresentada a metodologia e o cenário de avaliação considerados, para em seguida investigar o custo do TEE em termos de aumento de latência e diminuição do número de quadros capturados.

6.1. Metodologia de Experimentação

Dois experimentos distintos são conduzidos para validar a ferramenta proposta. Para avaliar a sobrecarga devido ao TEE, é desenvolvida uma versão da aplicação que executa integralmente no ambiente inseguro REE. Essa versão apresenta uma implementação equivalente à utilizada no ambiente seguro, diferindo apenas nas bibliotecas criptográficas empregadas, em função das particularidades dos algoritmos e APIs disponíveis no TEE. A partir deste ponto, a versão executada exclusivamente no REE é denominada *REE-sniffer*, enquanto a versão proposta que executa a lógica sensível no TEE é denominada *TA-sniffer*. Os resultados também foram comparados com a ferramenta *tcpdump*, comumente utilizada em *sniffers* e que não realiza anonimização dos dados.

Ambos os experimentos são conduzidos a partir de um mesmo cenário experimental em ambiente de laboratório. Para cada configuração avaliada, são realizados múltiplos processos de captura, nos quais um segundo Raspberry Pi, equipado com a mesma interface Wi-Fi, executa um programa em C atuando como transmissor, enviando pacotes na taxa de saturação, com MACs distintos. Todo o experimento é realizado no interior de uma caixa metálica fechada de forma a minimizar a interferência do tráfego Wi-Fi externo. Para minimizar ainda mais a influência do tráfego externo, a antena é retirada da interface do *sniffer*. As capturas têm duração de 1 minuto e são repetidas cinco vezes para cada tamanho de *batch* avaliado (1, 2, 4, 8, 16, 32, 64 e 128). Os testes são realizados tanto no modo *online*, no qual a anonimização ocorre em tempo real durante a captura; quanto no modo offline, no qual o processamento é realizado após a coleta dos pacotes. Assume-se um cenário de pior caso de tráfego, no qual o transmissor envia quadros *probe request* a uma taxa suficiente para saturar a primeira taxa de transmissão física básica definida para quadros de gerenciamento no IEEE 802.11, de 1 Mbps (quadros de gerenciamento são enviados em difusão, em uma das modulações básicas do padrão).

O primeiro experimento consiste na medição do atraso introduzido pelas operações de anonimização, incluindo geração de nomes temporários e substituição por pseudônimos finais. A latência por pacote é obtida contabilizando-se o tempo total gasto na execução dessas operações dentro do TEE, ou de suas contrapartes no REE, e dividindo-se esse valor pelo número total de pacotes processados. Adicionalmente, mede-se o tempo de transição entre REE e TEE, e vice-versa, que ocorre durante a chamada de funções com assinaturas idênticas às utilizadas na aplicação.

O segundo experimento tem como objetivo avaliar se a latência medida no primeiro experimento pode levar ao estouro da memória em cenários nos quais a taxa de chegada de pacotes supera a velocidade de processamento em tempo-real do *sniffer*, resultando em perda de quadros no arquivo de captura final. Para isso, a quantidade de pacotes recebidos e anonimizados com sucesso pelo *REE-sniffer* é utilizada como *baseline* e comparada com os resultados obtidos pelo *TA-sniffer*.

6.2. Experimento 1 – Latência

Os resultados do primeiro experimento indicam que a execução das operações de anonimização dentro do TEE introduz um aumento de entre 1758% e 1073% na latência por pacote quando comparada à execução no REE, observado em ambas as fases da aplicação. A Figura 2 apresenta os resultados de latência por pacote em função do tamanho do batch, e obtidos para as duas implementações.

No modo *online*, observa-se uma tendência de redução da latência à medida que o tamanho do vetor de entrada aumenta, com ganhos menores a partir de 8 quadros. Esse comportamento sugere que a estratégia de transferência em lote (*batches*) adotada oferece ganho de desempenho, amortizando o custo fixo das trocas de contexto entre o REE e o TEE quando múltiplos pacotes são processados em uma única chamada.

Durante a etapa *offline*, a latência absoluta por pacote é significativamente maior, de modo que o impacto relativo da troca de contexto torna-se desprezível. Assim, a tendência de redução de latência observada no modo online cessa. Nesse cenário, o aumento no tempo de processamento passa a ser associado ao uso do armazenamento seguro do TrustZone, no qual os dados são encriptados antes de serem escritos em disco. Assim, a sobrecarga é de aproximadamente 1100% para qualquer tamanho de batch.

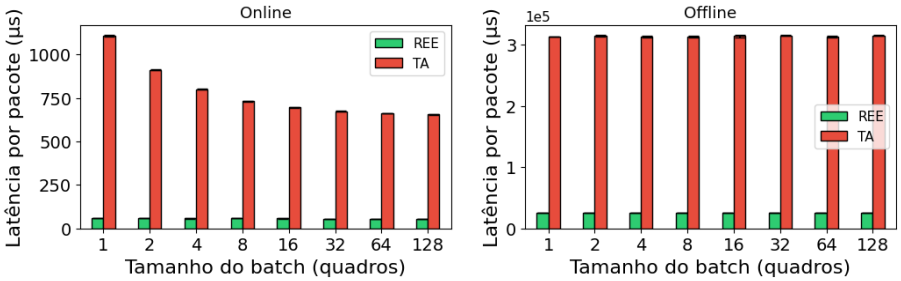

Figura 2. Latência média por quadro em função do tamanho do batch para as implementações no REE e no TEE, nas fases online e offline.

As duas principais fontes de sobrecarga introduzida pelo TEE são o tempo de processamento introduzido pelo processo de anonimização e o tempo para realizar a transição entre REE e TEE. A Figura 3 isola o tempo devido apenas à troca de contexto entre os mundos seguro e inseguro. Os resultados evidenciam um comportamento consistente com o observado na Figura 2 para o caso *online*, no qual a latência tende à estabilização a partir de vetores com aproximadamente 8 quadros. Isso reforça que a sobrecarga associada às entradas e saídas do TEE constitui um dos principais fatores no desempenho no modo *online*. A sobrecarga restante de latência se deve ao fato de que as operações criptográficas de anonimização realizadas no TEE são mais lentas do que as realizadas no REE.

6.3. Experimento 2 – Perda de Quadros

O segundo experimento avalia se o aumento de latência resulta em perda significativa de quadros durante a captura à taxa máxima. A Tabela 3 apresenta a quantidade média de pacotes processados por captura para o *REE-sniffer*, o *TA-sniffer* e o *tcpdump*, bem como a taxa de perda relativa ao *tcpdump*. Os resultados mostram que tanto o *REE-sniffer* quanto o *TA-sniffer* apresentam uma taxa de perda próxima a 6% em relação ao *tcpdump*, independentemente do tamanho do *batch* avaliado. Comparando diretamente o *REE-sniffer* e o *TA-sniffer*, observa-se que a diferença na quantidade de pacotes processados é marginal. Em alguns cenários, o *TA-sniffer* apresenta desempenho ligeiramente superior ao *REE-sniffer*, enquanto em outros ocorre o inverso, sem um padrão consistente que indique degradação de performance sistemática causada pelo uso do TEE.

Embora o processamento dentro do enclave introduza um aumento expressivo da latência por pacote, esse atraso não se traduz em perda adicional de quadros no arquivo de captura final. Na prática, o gargalo da captura permanece limitado pela taxa de recepção da interface Wi-Fi e pelo próprio subsistema de captura do Linux, e não pelas operações

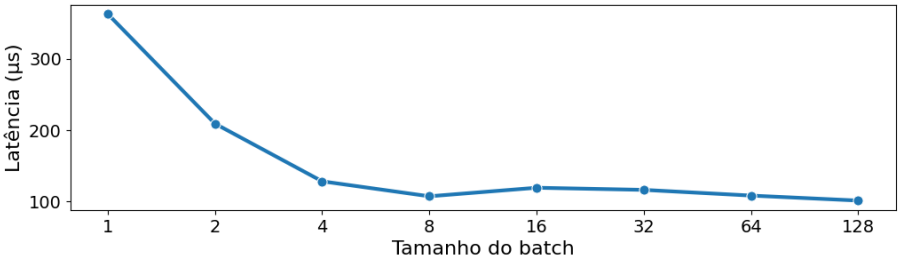

Figura 3. Impacto da troca de contexto entre o REE e o TEE na latência por quadro, em função do tamanho do batch.

Tabela 3. Número médio de quadros capturados e taxa de perda relativa em comparação com o tcpdump, para o *REE-sniffer* e o *TA-sniffer*. O tcpdump é apresentado como referência e não depende do tamanho do batch. Os intervalos correspondem a um intervalo de confiança de 95%.

	Quadros capturados			Perda relativa (%)	
Batch	REE	TA	Tcpdump	REE	TA
1	1075 ± 4	1071 ± 3	1143 ± 65	-5,91	-6,29
2	1074 ± 2	1077 ± 6		-6,03	-5,73
4	1074 ± 1	1072 ± 10		-6,05	-6,20
8	1076 ± 3	1072 ± 5		-5,87	-6,23

de anonimização executadas no ambiente seguro. A perda de quadros de 6% comparado com o tcpdump - que armazena as informações em claro - é considerada aceitável, considerando ainda que a situação é de pior caso, com saturação do enlace com quadros de gerenciamento Wi-Fi. Ou seja, para aplicações práticas de monitoramento Wi-Fi, a captura com o *TA-sniffer* executando em um dispositivo limitado é completamente viável.

7. Conclusões

Este trabalho apresentou um *sniffer* Wi-Fi com provisão de k-anonimato baseada em Ambientes de Execução Confiáveis (TEE), com foco na execução segura de operações de anonimização de endereços MAC e SSID. Diferente de trabalhos anteriores, a proposta: i) controla o valor de k-anonimato de forma determinística, ii) mantém a privacidade na presença de atacantes privilegiados, e iii) implementa otimizações para baixo consumo de memória. Os resultados experimentais mostram que a execução no TEE introduz sobrecarga em termos de latência quando comparada ao REE, especialmente na fase *offline*. Em termos de capacidade de captura, o uso do TEE não aumenta significativamente a taxa de perda de quadros, mesmo no cenário de taxa máxima de captura.

Os trabalhos futuros incluem o desenvolvimento de uma arquitetura distribuída de coleta coordenada entre TEEs, a implementação de um mecanismo de anonimização com k-anonimato multinível e a implantação de uma versão do *sniffer* em microcontroladores com arquitetura TrustZone-M.

Agradecimentos

O presente trabalho foi realizado com o apoio do CNPq (408255/2023-4), da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES), código de financiamento 001, da FAPERJ (E-26/204.122/2024), da FAPESP (2023/00673-7 e 2023/00811-0) e da Fundação de Desenvolvimento da Pesquisa - Fundep - Rota 2030.

Referências

Capponi, A., Fiandrino, C., Kantarci, B., Foschini, L., Kliazovich, D. e Bouvry, P. (2019). A Survey on Mobile Crowdsensing Systems: Challenges, Solutions and Opportunities. *IEEE Communications Surveys & Tutorials*, 21:2419–2465.

Das, A., Narayan, K. e Chakraborty, S. (2022). Leveraging ambient sensing for the estimation of curiosity-driven human crowd. Em *2022 IEEE International Systems Conference (SysCon)*, páginas 1–8.

Determe, J.-F., Azzagnuni, S., Singh, U., Horlin, F. e De Doncker, P. (2022). Monitoring Large Crowds With WiFi: A Privacy-Preserving Approach. *IEEE Systems Journal*, 16(2):2148–2159.

Göttel, C., Felber, P. e Schiavoni, V. (2019). Developing secure services for iot with op-tee: a first look at performance and usability. Em *Distributed Applications and Interoperable Systems: 19th IFIP WG 6.1 International Conference (DAIS)*, páginas 170–178.

Manjappa, R., Rao, V., Prasad, R. V. e Sinitsyn, A. (2019). Estimating crowd distribution using smart bulbs. Em *2019 IEEE Global Communications Conference (GLOBECOM)*, páginas 1–6. IEEE.

Martin, J., Mayberry, T., Donahue, C., Foppe, L., Brown, L., Riggins, C., Rye, E. e Brown, D. (2017). A study of mac address randomization in mobile devices and when it fails. *Proceedings on Privacy Enhancing Technologies*, 2017.

Ngabonziza, B., Martin, D., Bailey, A., Cho, H. e Martin, S. (2016). Trustzone explained: Architectural features and use cases. Em *IEEE 2nd International Conference on Collaboration and Internet Computing (CIC)*, páginas 445–451. IEEE.

Project, A. T. (2023). Teaclave trustzone sdk. `https://github.com/apache/incubator-teaclave-trustzone-sdk`. Acessado em: 2026-01-28.

Reichl, P., Oh, B., Ravitharan, R. e Stafford, M. (2018). Using Wifi Technologies to Count Passengers in Real-time around Rail Infrastructure. Em *International Conference on Intelligent Rail Transportation (ICIRT)*, páginas 1–5.

Ribeiro, R. H., Rodrigues, B. B., Killer, C., Baumann, L., Franco, M. F., Scheid, E. J. e Stiller, B. (2021). ASIMOV: A Fully Passive WiFi Device Tracking. Em *2021 IFIP Networking Conference (IFIP Networking)*, páginas 1–3.

Sakib, M. N., Halim, J. B. e Huang, C.-T. (2014). Determining Location and Movement Pattern Using Anonymized WiFi Access Point BSSID. Em *7th International Conference on Security Technology*, páginas 11–14.

Silva, F. D. M., kumar Mishra, A., Viana, A. C., Achir, N., Fladenmuller, A. e Costa, L. H. M. K. (2022). Performance Analysis of a Privacy-Preserving Frame Sniffer on a Raspberry Pi. Em *6th Cyber Security in Networking Conference (CSNet)*.

Tan, J. e Gary Chan, S.-H. (2021). Efficient Association of Wi-Fi Probe Requests under MAC Address Randomization. Em *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, páginas 1–10.

Tan, K. e Kotz, D. (2010). Saluki: A high-performance Wi-Fi sniffing program. Em *8th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks*, páginas 591–596.

Verma, V. e Singh, A. (2019). Indoor Location Determination using Radio Signal Strength Model for Distance Estimation. Em *2019 International Conference on Computer Communication and Informatics (ICCCI)*, páginas 1–4.

Wang, S., Zhang, L., Lin, L. e Zhang, L. (2021). Wi-Fi Sniffing Data Based Passenger-flow Detection Method Applied in Urban Rail Transit. Em *2021 IEEE 6th International Conference on Computer and Communication Systems (ICCCS)*, páginas 17–21.