Performance Analysis of a Privacy-Preserving Frame Sniffer on a Raspberry Pi

Fernando Dias de Mello Silva*, Abhishek Kumar Mishra†, Aline Carneiro Viana†, Nadjib Achir†‡, Anne Fladenmuller§, Luís Henrique M. K. Costa*

* *GTA/Poli/COPPE, Universidade Federal do Rio de Janeiro*, Rio de Janeiro, Brazil

† *Inria*, France

‡ *University Sorbonne Paris Nord*, Villetaneuse, France

§ *LIP6, Sorbonne Université*, Paris, France

Abstract—**Public Wi-Fi (IEEE 802.11) networks are an abundant data source that may serve different applications such as epidemic tracking and prevention, disaster response, crowdsensing, or ubiquitous urban services. Nevertheless, collecting and exploiting such data brings many privacy liabilities, considering that each transmitted frame has the MAC address (a unique device identifier) of the corresponding personal device, also considered sensitive information. Literature has shown that the MAC randomization performed by manufacturers of phones is not enough to protect devices' identification. Data obfuscation is a promising solution to avoid storing advertised identifiers of devices and prevent attackers from acquiring sensitive data. Obfuscating such identifiers while also being able to differentiate frames sent by different devices poses a significant challenge for frame capturing by low-resource IoT devices in real-time. This paper illustrates the impact of on-the-fly hashing as an obfuscating measure to protect people's privacy. Since no popular off-the-shelf sniffer (using `wireshark` or `tcpdump`) allows for on-the-fly hashing, we build upon the `scapy` library a custom-made sniffer capable of hashing the required data needed to protect user privacy. We demonstrate the viability of this privacy-preserving IoT sniffer on a Raspberry Pi platform.**

Index Terms—**privacy, anonymization, sniffer, hashing, IoT.**

I. INTRODUCTION

Passive frame sniffing has become an effective source of information given the widespread adoption of mobile devices with IEEE 802.11 capabilities, from smartphones to IoT (Internet of Things) devices. The passively collected data has multiple applications such as predicting user-mobility, contact tracing, and opportunistic networks [5], [6], [12], [14], [17], [22], [26]. Recently, passive sniffing has become easier through low-power microcomputer units, which enable entire operating systems to run on embedded devices, while allowing rapid development with tools initially made for desktop environments. Such sniffing flexibility enables developers to propose scalable solutions that allow multiple sniffers to acquire data from large regions and have a good sense of the medium being monitored.

Nevertheless, passive sniffing development is limited by data-protecting legislation that requires such devices to create a sniffing solution with obfuscation techniques to protect the user's privacy. The legislation exists due to the demonstration of various kinds of attacks from sniffed wireless packets in the literature [4], [16], [19].

Sniffers are vulnerable to a variety of attacks which can be broadly classified into compromising (i) *confidentiality*, (ii) *data protection*, (iii) *integrity*, iv) *availability* and (v) *access control* [3], [23]. We focus on the first three types of privacy breaches in this work, as they are the most devastating from the user's perspective.

We introduce a *promising and particularly challenging objective* to circumvent the breaches mentioned above: *on-the-fly obfuscation*. A sniffing process interfaces with the wireless network interface card (NIC) for frame acquisition. For each captured frame, the sniffing process anonymizes and saves it in the disk storage of the sniffer. Hence, on-the-fly obfuscation brings the benefits of reducing consequences of possible attacks to the sniffer. We illustrate the collection process for the sniffed public Wi-Fi packets in Figure 1. First, the user data is protected at the end of the capturing process inside the sniffer (represented by a Raspberry Pi in the figure). Second, any sensitive data leakage happening during the transfer from the sniffer RAM to its disk will never compromise the rest of the data previously stored at the disk.

In opposition to the aforementioned on-the-fly sniffing proposal, a regular data-network packet capturing and analyser tool, such as `tcpdump` [1] directly store captured data in its raw format at the sniffer disk, without any previous data protection. Posterior protection, as for hashing anonymization, has to be performed separately at the sniffer hard disk after the raw data storage. In case of a malicious attack, this post-storage protection processing clearly brings vulnerability to passive sniffing strategies and to collected suers data.

However, on-the-fly obfuscation poses a limitation to constrained devices: The more complex the obfuscation technique is, the more frames passive sniffers might lose when their CPU time is dedicated to frames' anonimisation processing than reading. This limitation affects the collected data resolution and quality. To measure and understand related limitations, we
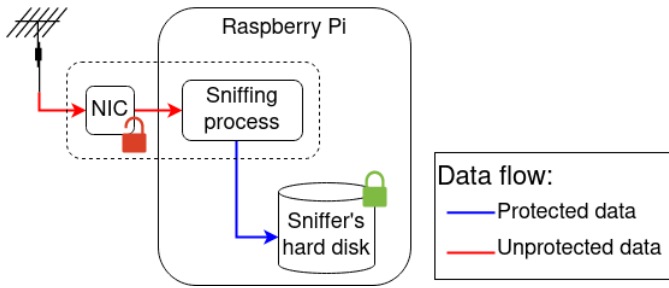
Fig. 1: Diagram of the data flow in the sniffing process for a privacy-preserving sniffer. Data is given unprotected by the NIC driver to the sniffing process that secures the data and saves it at the sniffer hard disk.
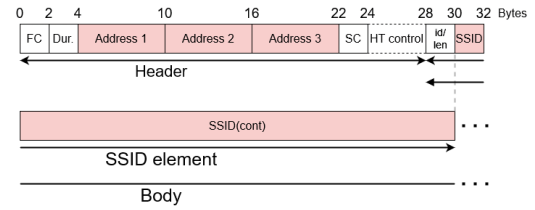

Fig. 2: Format of a MAC management frame header with privacy concerning fields highlighted in red with sizes in bytes. Optional fields are represented with dotted lines.

analyze the impact of an obfuscation technique that consists of hashing the sensitive data transmitted by public Wi-Fi packets.

We analyze the performance of on-the-fly obfuscation techniques using a Raspberry Pi 4 model B (Pi) device, refered hereafter as privacy-preserving sniffer. The objective is to assess this single-board computer's capacity to capture, process, and obfuscate public Wi-Fi probe-requests. Indeed, public Wi-Fi probe-requests are considered sensible data and useful in tracking applications. In this context, we use as benchmark, the previously mentioned state-of-the-art packet capturing tool, the `tcpdump`, which does the packet sniffing without any obfuscation. Then, we compare the performance of a privacy-preserving sniffer with the benchmark one under different modes of obfuscation.

The main contributions of this paper are:

- We introduce the on-the-fly obfuscation on constrained sniffing devices as an alternative for more secure handling of sniffed privacy-sensitive data.
- We perform a thorough analysis of the impact of each privacy algorithm on the Pi's ability to process packets when imposing the on-the-fly obfuscation. The analysis provides a discussion on the features and trade-offs of such sniffers in resource-constrained devices.
- We provide in-depth insights on the resource utilization of Pis during the entire capturing process of a privacy-preserving sniffer.
- Upon publication of this work, we intend to release the *on-the-fly sniffer* tool as an open-source solution to on-the-fly obfuscation of public Wi-Fi frames.

This paper is organized as follows. Section II briefly introduces the scanning process that generates the data to be obfuscated. Section III discusses the methodology to measure the performance of a Raspberry Pi when hashing. Section IV discusses the results and corresponding insights. Section V presents related work on packet sniffers and user privacy. Section VI concludes the paper and presents future directions.

II. Scanning in WiFi

The IEEE 802.11 protocol [13, p. 821] defines scanning as a process for discovering nearby access points by host devices. The process can be active or passive. In active scanning, devices transmit frames asking for the identity, connection, and compatibility information from all nearby Access Points (AP). In passive scanning, a device just eavesdrops for beacon frames transmitted by the AP.

Both methods use management frames which contain three MAC address fields that could compromise user privacy, as seen in Figure 2. Some of those frames may also carry information about the Service Set Identifier (SSID) related to a particular access point (AP). The rest of the frame is composed of numerous frame body elements which hold information about multiple aspects of the protocol, the device, and even arbitrary data that is specific to the device's manufacturer.

The passive scanning consists of an AP transmitting at fixed intervals, a *beacon frame* that contains its MAC address, SSID, and other relevant information about this particular AP. This information could be used to estimate the sniffer's location based on the surrounding SSIDs and prior knowledge of their geographical locations. This beacon frame is sniffed by applications that keep a database of geographical locations of APs. Even with significant relevance, this data is associated with fixed APs most of the time and doesn't disclose any information about users present in the same region.

In this paper, we focus on active scanning frames since those frames contain the potential relevant user-location tracking information and also sensitive data.

A. Active scans

The active scanning procedure consists of a user's device emitting a *probe request* frame, asking all nearby APs to respond with *probe responses* with their respective MAC addresses and SSID values. The frame could be a wildcard probe request which requests the response of all of the APs nearby or contains the specific SSID and the MAC address of a particular AP known to the user to limit the search of available connections.

B. Privacy threats

A user's device usually sends its true MAC address in the source address field of the *probe request* and often sends along with the *known* SSIDs. Some devices send randomised MAC addresses, but even then, they could be tracked [24]. The SSIDs can be associated with places regularly attended by the user (her house, workplace, and so on). This reveals that it is not only the user device that can be identified from the
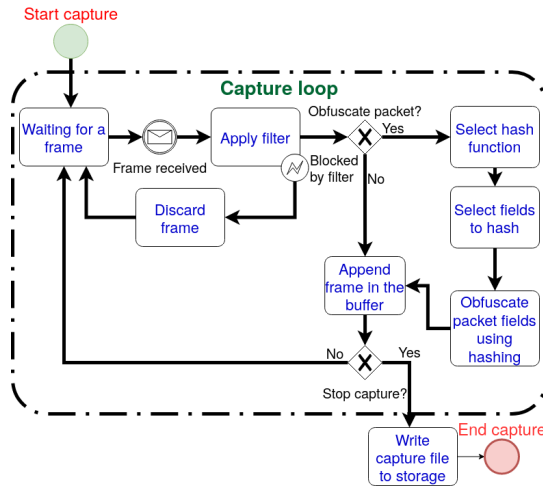
Fig. 3: Design of the privacy-preserving sniffer.

probe request but also the previous networks it has been connected to. Suppose publicly available databases containing the geographical location of publicly available SSIDs are utilized. In that case, it is possible to infer geographical information about a user just by passively sniffing probe requests, posing a significant privacy leak. [9] extracts geographical information from probe requests to match people in a crowd with their places of origin.

Given the high occurrence that such frames could reach in some areas [11] and also the relevance of the information it carries, it is necessary to create a secure method for obfuscation of those values without compromising the ability to distinguish between frames sent by different users.

As explained in Section I, the introduced on-the-fly sniffer addresses the issue of compromising user-sensitive data by obfuscating it before dumping it in the sniffer's local memory. We consider the MAC address and one SSID field when analyzing the impact of obfuscation in on-the-fly processing.

III. REALISING A PRIVACY-PRESERVING SNIFFER

In this section, we first detail the proposed privacy-preserving sniffer. Next, we describe the construction of the benchmark sniffer. We then present the experimental setup and describe the different configurations used for our experiments.

A. Privacy preserving sniffer

The privacy-oriented on-the-fly sniffer is a module capable of being efficiently deployed over a Pi. We present its design and workflow in Figure 3. The sniffer stays in a loop and, for each received packet, checks if the packet is a probe request using a filter and then, if valid, applies the obfuscation to the desired fields. Hashing is the obfuscation technique we choose in this paper.

We develop the sniffer on top of scapy [2], an open-source tool for creating sniffers, injectors, and general tasks of packet manipulation. We choose scapy as it provides a ready-made interface to interact with the sniffer functionality of network cards at a high level. It allows us to introduce

specific functionalities using packet contents each time a new packet is received.

In the sniffer, for each received frame, we hash received address fields and truncate its value to the same length as the original MAC address (48 bits) before saving it to the disk (hence on-the-fly). The truncation after hashing MAC addresses is a way to purposefully introduce collisions and therefore mitigate the issues pointed out in [8]. We hash and truncate SSIDs to 48 bits as well.

Once we complete the obfuscation, the sniffer saves the current frame in a PCAP file format to the buffer. In this work, we consider two different hashing algorithms (SHA256 [20] and MD5 [21]) while varying the frame fields to obfuscate. The two algorithms were selected for their efficiency in running in a lightweight system like the Pi.

Though, in the sniffer's design, we let the user have the flexibility of specifying any hash algorithm and the set of packet fields. Moreover, the sniffer's current implementation allows further optimization in the future using techniques like multi-threading and the periodically use hash-tables to save the obfuscated MAC addresses. to avoid repeated calculations

B. Benchmark sniffer

We consider the performance-benchmark sniffer as the sniffer utilizing the state-of-the-art packet capturing tool: tcpdump. It allows sniffing on a selected interface and dumps the captured data in the same PCAP format on Pi's disk. However, there is no anonymization of frame fields in the sniffed data. Moreover, this sniffer has limited functionalities regarding frame-filter configurations, trace file options, and truncation of the captured frame while storing sniffed frames.

Although, due to its implementation in C language and the use of the libpcap [1] to provide an Application Programming Interface (API) to a selected NIC for sniffing, it is expected to perform better than our proposed on-the-fly obfuscation sniffer. However, this comes at the cost of neglecting user privacy, which makes it a perfect choice as a benchmark sniffer to investigate the degree of privacy vs. the performance trade-off.

C. Experimental setup

We illustrate the experimental setup in Figure 4. We use two Raspberry Pi Model 4B single-board computers for the experiments, equipped with external TP-Link TL-WN722N v3.2 wireless interfaces. We place the sniffers in an enclosed metal container with only a single open passage for power and Ethernet cables. One Raspberry Pi acts as a transmitter to send Wi-Fi frames, while the other Pi acts as a receiver that captures frames in monitor mode and stores them. We control the Pis using another machine with built-in scripts to configure each device and control all aspects of the test procedure. Also, a USB extension links another wireless interface inside the metal box to a desktop computer outside. The desktop sends the respective commands to the Pis and sniffs the environment inside the box.
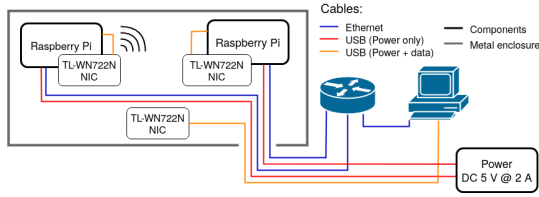
Fig. 4: Diagram of the setup and wired connections.

ID	Sniffer	Fields	Hash
A	`tcpdump`	-	-
B		No Fields	None
C		Addresses	MD5
D	`scapy`	Addresses	SHA256
E		Addresses + SSID	MD5
F		Addresses + SSID	SHA256

TABLE I: List of experimental configuration identifications.

While running the experiments, we deploy another Python process run in parallel to collect the resource usage information at the receiver node. The process saves the resource information every 3 seconds and it does so throughout the entire capture for each test configuration.

We develop a custom transmitter module written in C language utilising `libpcap` [1]. The transmitter reads from a PCAP capture file and uses the timestamp information of each contained frame to time the frame injection into the NIC of choice. Its function is to replicate the environment captured by the PCAP files by sending the frames within with the same delay between them as they were received. We evaluate the performance of the transmitter in Section IV-A.

We synthetically generate the PCAP files selected for frame transmission for our experiments. We use `scapy` and its packet creation and manipulation functionalities to realize the PCAP generation module. This module creates probe request capture files with flexible inter-frame delays, the size of the SSID elements, and the size of frame payloads.

D. Experimental scenarios

Each experiment consists of transmitting a specific synthetically generated capture file while running a sniffer with a specific set of configurations. The output of an experiment is a PCAP capture file from the receiver, a resource graph with CPU and memory utilisation, CPU temperature, and, frame transmission information to calculate the average throughputs in kbps and frames per second. The receiver's capture file is used to measure the receiver's throughput by slicing it in 10 seconds intervals and averaging the results for each slice to obtain statistical results for the whole experiment.

In total, we transmit 10 separate capture files that were created synthetically using the file creator mentioned in Section III-C. All of the files only contain probe requests with arbitrary values for all of the fields without any time delay between frames, to potentially saturate the channel. The frame body elements are variable, and it is not possible to determine a fixed probe request size. Two test groups are considered: One with the smallest probe request size possible and the other with the largest one.

To emulate a larger-sized probe request, we append extra bytes after the SSID element, limited to a maximum of 1024 extra bytes. In this paper, we denote these extra bytes in probe requests as "payload." For each experiment group with and without payload, we vary the size of SSID between 0, 8, 16, 24, and 32 bytes. Finally, the number of transmitted frames is chosen considering the corresponding frame size (determined

by the SSID and the payload), to homogenise all the test duration to around 3 minutes (cf. Section IV-A).

We list all experimental configurations in Table I. For each configuration, we define an identification (ID) to label experimental results (A - F). For all the configurations, we set both sniffers to channel 1 and Berkeley Packet Filters (BPF) to capture only probe requests. For the on-the-fly sniffer, we configure it to capture and store directly without hashing as well as after hashing with MD5 and SHA256. When hashing is enabled, we test two hashing possibilities: hashing addresses only and hashing addresses and the SSID value.

Also, we execute a third sniffer on the desktop, which confirms that packets sent by the transmitter were actually being transmitted over the medium and not being dropped by the operating system of the Pi.

IV. RESULTS

In this section, we first show the results regarding the behaviour of the transmitter module. Next, we compare the throughput of the privacy-preserving and the benchmark sniffer. Then, we show the impact of the hashing in the on-the-fly sniffer. Finally, we illustrate the overall resource utilisation by sniffers.

For the box plots in this section (Figures 5 through 9), we limit the whiskers by 1.5 times the interquartile range (except on figure 9, where they limit the maximum and minimum values). Also, orange bars represent median and green triangles average values. When no whiskers or boxes of a column appear, it signifies that the variation was not enough to appear in the plot with the used resolution.

A. Frame transmission

We observe the transmitter throughput with respect to the variation in the size of SSID contained in each frame, in Figure 5. The grey markers represent the transmission with an extra payload appended to the end of each frame and the orange markers represent the transmission without the extra payload. Each pair from those different groups are aligned with the X-axis column that displays their selected size for the SSID field.

Figure 5a shows that the minimum throughput obtained between all transmission types is over 725 kbps from the files without payload whereas the maximum reaches 1 Mbps. The variation in throughput from 1 Mbps is due to imprecision in the compensation method of adjusting the frames per second rate to match the maximum throughput available. The

management frames are transmitted at base rates [13] and for our selected version (802.11b), the chosen base rate is 1 Mbps. This validates the correctness of our transmitter.

This compensation is demonstrated when comparing the y axis of the graphs in Figure 5b. For both graphs, it is expected that the frames per second rate decrease with larger frames. Since the files without payload have a frame size significantly lower, the rate in frames per second has to be one order of magnitude higher to keep the bit rate relatively constant and the tests at the same duration.
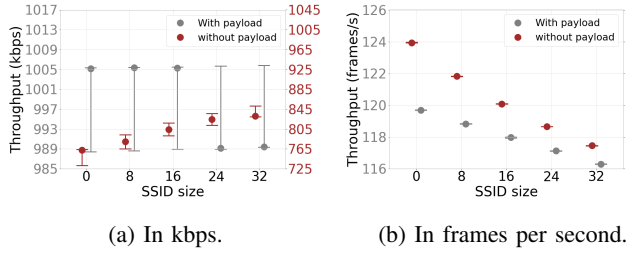
(a) In kbps.

(b) In frames per second.

Fig. 5: Transmission average throughput results for all files and separated between files with and without extra payload.

B. Frame reception in a privacy-preserving sniffer

We see a relevant characteristic of the receiver's throughput in Figure 6. In the case without payload, where the throughput is around 2,000 frames per second, `tcpdump` majorly outperforms the other configurations by at least 12 times more, as seen in Figure 6a. When inspecting the same graph without the `tcpdump` column (ID A in Table I), it is possible to see that the maximum throughput achieved by the module does not pass 140 frames per second, as seen in Figure 6b.

The graphs in Figure 8 confirm that this limit is not a data rate constraint since that with payload the on-the-fly sniffer can reach very close to the maximum throughput (as seen in Figure 8b), which indicates an underlying problem with packet creation that limits its processing considerably. Although this is a considerably smaller rate than the `tcpdump` original alternative, it is still larger than the average rates of probe requests per second found in [11]. Therefore, this method would still be sufficient in a real-world scenario with the majority of frames being probe requests.

The throughput measured in kbps can be seen in Figure 8. Figure 8a shows the impact of the frame per second limitation on the throughput. Since the limitation is observed when dealing with frames per second, large frames (i.e. the ones with the extra bytes of payload) get processed at the same rate and therefore the apparent throughput appears higher. Both graphs from Figure 8b show the throughput in kbps of all of the configurations with only the payload.

The graph on the left displays the statistical throughput for intervals of 10 seconds while the graph on the right displays the total average of the entire transmission run. The purpose of the graph on the right is to assure stability between tests and it shows that the transmitter side kept stable throughout the different sniffer configurations varying only about 2% between

(a) All configurations.

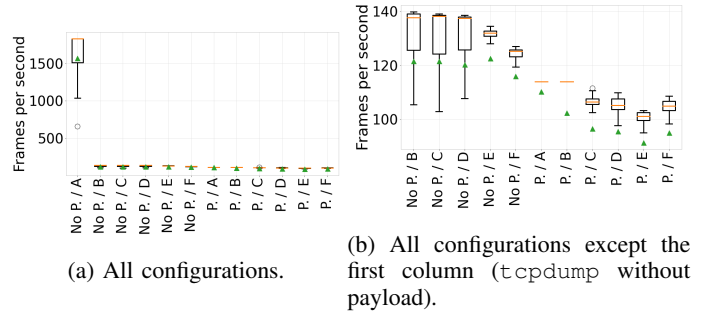(b) All configurations except the first column (`tcpdump` without payload).

Fig. 6: Receiver's throughput for SSID of size 0 of different configurations.

tests. When compared to the receiver side, it's possible to see that without any obfuscation functionality the Raspberry was able to follow the transmission's throughput while keeping the throughput stable. The considerably lower value for average is due to outliers that are not shown at this scale.

C. Impact of hashing on the on-the-fly sniffer

The efficiency of the hashing functions is better analysed in Figure 7. The figure shows the throughput in the same way as the graph on the left of Figure 8b, but now they appear in frames per second and three graphs now illustrate the impact of different SSID lengths, with and without payload.

The three graphs show the following patterns: When no hashing function is applied (ID B), the throughput tends to be higher by median except for Figure 7a when there's no payload. Between different hashing algorithms either with (IDs E and F) or without (IDs C and D) payload, the addresses don't display any significant changes except for Figure 7b where the SHA256 algorithm (selected in IDs D and F) has better results than the MD5 algorithm.

Finally, except for the graph in Figure 7a, losses in frames per second when the hashing was enabled were in the best case scenarios of about 6,7% when not hashing the SSID and 10% when doing so. This comparison indicates that in some cases the on-the-fly sniffer had a noticeable impact on its performance when it applied hashing functions to the frames.

D. Overall resource utilisation

The resource usage all followed the same pattern between SSID values. The CPU usage was 100% for all configurations using the on-the-fly sniffer, as seen in Figure 9a. Noticeably `tcpdump` had little to no impact on the CPU usage. The higher CPU usage is expected from the on-the-fly sniffer, given the per-packet obfuscation that it performs.

The memory usage varied but never grew over 15 MB, the average size of the synthetically created capture files, as can be seen in Figure 9b. Memory consumed less space without the payload but its consumption noticeably increased the more the SSID size increased. The temperature of the PI was kept constant at around 75° throughout the experiment. The resource plots for one SSID value can be seen in Figure 9.
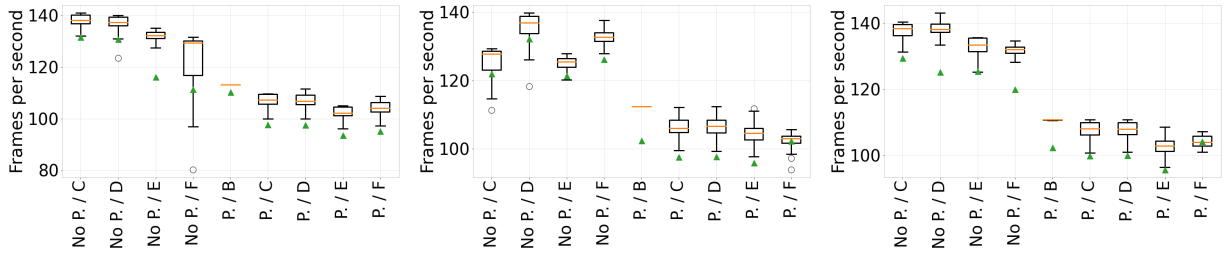
(a) Throughput for SSIDs with 8 bytes.   (b) Throughput for SSIDs with 16 bytes.   (c) Throughput for SSIDs with 32 bytes.

Fig. 7: Receiver's throughput for all of the on-the-fly sniffer configurations, with the SSID size varying in 8, 16 and 32 bytes.
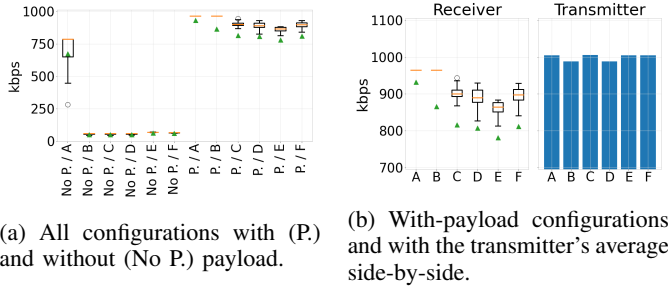

(a) All configurations with (P.) and without (No P.) payload.

(b) With-payload configurations and with the transmitter's average side-by-side.

Fig. 8: Receiver's throughput for SSID of size 0.
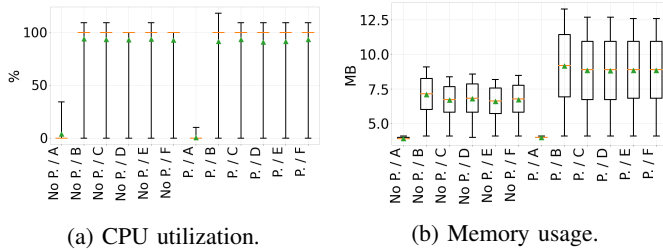

(a) CPU utilization.

(b) Memory usage.

Fig. 9: Receiver's resource monitoring for different configurations with 16-byte long SSID.

V. RELATED WORK

In regards to the privacy liabilities involving probe requests, numerous studies point out the issues with the current standard and various ways of exploiting them. In [18], it is illustrated multiple attacks that circumvent MAC address randomisation (a technique done by users devices that consists of attributing a new random address to protect one's real MAC address) by timing attacks and inferences. [9] uses the probe requests information to leverage information about crowds regarding their region of origin and plurality, which became an indirect measurement for an election prediction. [10] has discussions about scenarios in which possible and more personal attacks can be performed with this sort of data. This paper considers those issues as a necessity to create obfuscation solutions for sniffing.

The hashing method, as pointed out in [8], [25] has its drawbacks in regards to privacy. It's known that the range of MAC addresses is limited by manufacturers and this poses

an advantage for a brute force approach in an attempt to de-obfuscate the frames. The implementation detail needed to circumvent this issue is considered in our work. As seen in [7], those are not the only desired characteristics in a wifi-based tracking application that make them privacy-oriented, and for IoT devices that implies significant more overhead than what is observed in our work.

The availability of probe requests in the real-world scenario is discussed in [11]. In their work, sniffing is done in a stadium to measure the number of probe requests made in a crowded environment without any access points nearby. They find that the maximum throughput measured is approximately 62 frames per minute. Our work shows that this number is well within the range of all of the sniffing approaches, and the analysis considered involved idealistic and worst-case scenarios to identify where are the bottlenecks.

There have been efforts in the literature to evaluate the performance aspects of a WiFi tracking system using IoT devices. In [15], factors of the capture process such as the operating channel, the signal strength of nearby access points and the number of devices in the vicinity are all considered to evaluate the performance of a sniffing solution. However, the work lacks consideration with regard to the privacy of collected data.

VI. CONCLUSION

This paper analyses the privacy vs the performance trade-off when using the introduced privacy-preserving sniffer, which uses on-the-fly obfuscation. The traditional sniffer outperforms ours' rate by 12 times, but our approach processes frames at a rate above what was observed in a crowded scenario [11]. The obfuscation techniques decreased the sniffers' performance as expected, but the results were only constant when comparing the number of fields to be hashed, and performance differences over the hashing algorithm used were inconclusive. This demonstrates the promise of the on-the-fly sniffer specifically in non-saturated wireless conditions. The traditional approach has an advantage in resource consumption and may impact the decision between the two studied options.

More advanced obfuscation techniques could be explored and its impact on a Raspberry Pi could be analysed. The optimisations mentioned in Section III-A can be investigated in future versions of the on-the-fly sniffer.

REFERENCES

[1] Tcpdump. Available at: https://www.tcpdump.org/. Last access in 18/08/2022.

[2] Scapy. Available at: https://scapy.net/. Last access in 18/08/2022.

[3] Wiem Bekri, Taher Layeb, Rihab Jmal, and Lamia Chaari Fourati. Intelligent IoT systems: security issues, attacks, and countermeasures. In *2022 International Wireless Communications and Mobile Computing (IWCMC)*, pages 231–236, 2022.

[4] Tomas Bravenec, Joaquín Torres-Sospedra, Michael Gould, and Tomas Fryza. Exploration of user privacy in 802.11 probe requests with MAC address randomization using temporal pattern analysis, 2022.

[5] Kwon Nung Choi, Thilini Dahanayaka, David Kennedy, Kanchana Thilakarathna, Suranga Seneviratne, Salil S. Kanhere, and Prasant Mohapatra. Poster abstract: Passive activity classification of smart homes through wireless packet sniffing. In *2020 19th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, pages 347–348, 2020.

[6] Anirban Das, Kartik Narayan, and Suchetana Chakraborty. Leveraging ambient sensing for the estimation of curiosity-driven human crowd. In *2022 IEEE International Systems Conference (SysCon)*, pages 1–8, 2022.

[7] Levent Demir, Mathieu Cunche, and Cédric Lauradoux. Analysing the privacy policies of wi-fi trackers. In *Proceedings of the 2014 workshop on physical analytics - WPA '14*, pages 39–44. ACM Press.

[8] Levent Demir, Amrit Kumar, Mathieu Cunche, and Cedric Lauradoux. The pitfalls of hashing for privacy. 20(1):551–565.

[9] Adriano Di Luzio, Alessandro Mei, and Julinda Stefa. Mind your probes: De-anonymization of large crowds through smartphone WiFi probe requests. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9, April 2016.

[10] Ben Greenstein, Ramakrishna Gummadi, Jeffrey Pang, Mike Y Chen, Tadayoshi Kohno, Srinivasan Seshan, and David Wetherall. Can Ferris Bueller still have his day off? Protecting privacy in the wireless era. In *11th Workshop on Hot Topics in Operating Systems (HotOS)*, May 2007.

[11] Xueheng Hu, Lixing Song, Dirk Van Bruggen, and Aaron Striegel. Is there WiFi yet? How aggressive WiFi probe requests deteriorate energy and throughput. (arXiv:1502.01222), February 2015. arXiv:1502.01222 [cs].

[12] Baoqi Huang, Guoqiang Mao, Yong Qin, and Yun Wei. Pedestrian flow estimation through passive WiFi sensing. *IEEE Transactions on Mobile Computing*, 20(4):1529–1542, 2021.

[13] IEEE. IEEE standard for information technology–telecommunications and information exchange between systems - local and metropolitan area networks–specific requirements - part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. pages 1–4379. Conference Name: IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016).

[14] Zann Koh, Yuren Zhou, Billy Pik Lik Lau, Chau Yuen, Bige Tuncer, and Keng Hua Chong. Multiple-perspective clustering of passive Wi-Fi sensing trajectory data. *IEEE Transactions on Big Data*, pages 1–1, 2020.

[15] Yan Li, Johan Barthelemy, Shuai Sun, Pascal Perez, and Bill Moran. A case study of WiFi sniffing performance evaluation. *IEEE Access*, 8:129224–129235, 2020.

[16] Edoardo Longo, Alessandro E. C. Redondi, and Matteo Cesana. Pairing wi-fi and bluetooth mac addresses through passive packet capture. In *2018 17th Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net)*, pages 1–4, 2018.

[17] R. Manjappa, V. Rao, R. V. Prasad, and A. Sinitsyn. Estimating crowd distribution using smart bulbs. In *2019 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2019.

[18] Abhishek Kumar Mishra, Aline Carneiro Viana, Nadjib Achir, and Catuscia Palamidessi. Public wireless packets anonymously hurt you. In *2021 IEEE 46th Conference on Local Computer Networks (LCN)*, pages 649–652. IEEE.

[19] Aynaz Nazerian and Faride Parastar. Passive iot device fingerprinting using wifi. In *Proceedings of the 12th ACM Wireless of the Students, by the Students, and for the Students (S3) Workshop*, S3 '21, page 6–7, New York, NY, USA, 2021. ACM.

[20] U.S. Department of Commerce, National Institute of Standards, and Technology. *Secure Hash Standard - SHS: Federal Information Processing Standards Publication 180-4*. CreateSpace Independent Publishing Platform, North Charleston, SC, USA, 2012.

[21] R. Rivest. *RFC1321: The MD5 Message-Digest Algorithm*. RFC Editor, USA, 1992.

[22] Ahmed Sheikh. *Sniffers and Social Engineering*. Apress, Berkeley, CA, 2021.

[23] Preeti Sinha, Amit kumar Rai, and Bharat Bhushan. Information security threats and attacks with conceivable counteraction. In *2019 2nd International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICICT)*, volume 1, pages 1208–1213, 2019.

[24] Jiajie Tan and S-H Gary Chan. Efficient association of wi-fi probe requests under mac address randomization. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*, pages 1–10. IEEE, 2021.

[25] Mathy Vanhoef, Célestin Matte, Mathieu Cunche, Leonardo S. Cardoso, and Frank Piessens. Why MAC address randomization is not enough: An analysis of Wi-Fi network discovery mechanisms. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, page 413–424, Xi'an China, May 2016. ACM.

[26] Edwin Vattapparamban, Bekir Sait Çiftler, Ismail Güvenç, Kemal Akkaya, and Abdullah Kadri. Indoor occupancy tracking in smart buildings using passive sniffing of probe requests. In *2016 IEEE International Conference on Communications Workshops (ICC)*, pages 38–44. IEEE, 2016.