Obfuscating Patterns of Life with Trusted Execution Environments

Jack Blackburn
Dept. of Computer Science
University of Colorado Boulder
Boulder, USA
john.blackburn@colorado.edu

Stefan Tschimben
Dept. of Computer Science
University of Colorado Boulder
Boulder, USA
stefan.tschimben@colorado.edu

Eric Keller
Dept. of Elect., Comp. and Energy Eng.
University of Colorado Boulder
Boulder, USA
eric.keller@colorado.edu

Tamara Silbergleit Lehman
Dept. of Elect., Comp. and Energy Eng.
University of Colorado Boulder
Boulder, USA
tamara.lehman@colorado.edu

***Abstract*—Network traffic, even when encrypted, can often reveal sensitive information. This work looks to obfuscate one person's pattern of life by recreating network traffic activity of multiple users on one device. We leverage existing technology, Trusted Execution Environments, in which the hardware can enforce isolation of the executing environment as well as provide protection from physical attacks by securing data in memory. We demonstrate a proof of concept that can model a persona's network traffic behavior from inside the Trusted Execution Environment using Intel SGX technology.**

***Index Terms*—Network traffic analysis, secure communications, trusted execution environment, secure hardware, secure memory**

I. Introduction

As 5G is becoming near-ubiquitous around the world, many organizations plan to take advantage of 5G's greater emphasis on security compared to previous generations of cellular communications. Taking advantage of indigenous 5G networks in foreign countries has the potential to eliminate the costs of installing and maintaining an alternate communications infrastructure. However, these 5G networks are deployed and operated by organizations that are untrusted and potentially hostile to the U.S. or humanitarian organizations in many areas of the world. Network operators can observe most activities over these untrusted networks and gather insights into the distinguishable online behavior of individuals relying on the network. Recent examples such as Russian officers being identified by their cellular connection during the war in Ukraine [1], or Polish troops revealing their exact location by using dating applications on their phones [2], highlight the need for new security technologies to enable secure operations in these environments.

While common intrusion methods like port-based monitoring and deep packet inspection (DPI) can be prevented by using random port numbers and encrypting traffic, these examples highlight that encryption alone is no longer enough to hide activity. As a network operator with access to encrypted user traffic, various clustering and machine learning techniques can be used to not only identify a user's applications, but also their operating system, and the user's online actions [3]–[5]. Additionally, as increasing amounts of data become available through the social media activities of individuals, various pattern analysis methods can be used to extract unique profiles. Analysis of social media platforms for example can be used to provide near-real-time information about events such as natural disasters and others [6], [7]. Together, these analysis methods can be used to create a unique and distinguishable profile that can be used to track a user's daily behavior.

We propose obfuscating this pattern of life analysis by generating synthetic traffic to create fake personas or shape existing ones. This work is based on the observation that by modeling known user activity mathematically we can obfuscate a user's real pattern. Recent research suggests multiple methods to obfuscate IoT traffic, among others using fixed-value obfuscation and TCP multipath transmissions [8]–[12].

Generating personas relies on the secrecy of the models of personas and the operational data in software on the plan to generate personas. As such, our system leverages secure hardware to protect the software and data. Trusted execution environments (TEE), such as ARM TrustZone and Intel Software Guard Extensions (SGX), represent an important first step towards securing a user's device against these types of attacks [13], [14]. TEEs are usually paired with secure memory features to protect the confidentiality and integrity of the data. Both Trustzone and SGX have been used in varied applications, including obfuscation of the control flow of a program by inserting fake statements [15], securing of the file system by encrypting and decrypting data inside the TEE [16], and enhancing the security and privacy of Tor networks [17].

In this work we are interested in using the security benefits of Intel SGX to hide network traffic and protect the obfusca-

This research was supported in part by the NSF under Grant No. 2226426

tion logic inside the TEE. Specifically, we make the following contributions:

- Propose a mechanism in which a person's traffic activity can be obfuscated.
- Perform the network injection work inside a TEE.
- Effectively hide a person's network traffic patterns.
- Develop a proof of concept to obfuscate patterns of life.

II. Motivation and Threat Model

The broad goal of an adversary is to perform pattern of life analysis on an individual (or group of individuals) - where they were, who they were with, and what were they doing. This may serve to help identify certain individuals and can generally help provide information useful for some objective - e.g., launch an attack, kidnap someone, be alerted to an attack.

The Adversary and their Capabilities. We assume an adversary with an ability to tap network traffic at an access point (e.g., 5G base station). This is likely in the form of a state-sponsored actor. The network access point will have visibility into the link layer identifiers of the devices associated with that access point. These identifiers, e.g., MAC address in 802.11, are included in the network traffic. With this, a direct tie between device and network traffic is established.

We assume an adversary cannot decrypt encrypted traffic. But, we do assume an adversary has ability to analyze traffic and can identify the services used. The packet size distribution, inter-packet timing, and nature of the interaction between client and server have all been used to demonstrate an ability to fingerprint network traffic.

With this, what can the adversary do? Given an ability to tie network traffic to a specific device and determine services being used, this can be quite possible. If we assume a device is being used by a single individual (which is likely for traffic from wireless connections such as a smart phone), an adversary can then perform pattern of life analysis on an individual (and, by extension, a group). That is, an adversary can track an individual's location, and can associate different profiles of network usage with activities (e.g., leisure vs preparing for an operation). Doing this over some time period is important information as part of pattern of life analysis.

Persona Tracking is more Powerful. To avoid being tracked, an individual may use different devices. In fact, they may even use multiple devices in the normal course of their day, without even attempting to avoid being tracked. Regardless of intent, through the use of different devices, tracking each device separately is insufficient to perform a useful pattern of life analysis on an individual.

What the adversary needs to do instead is to be able to associate network traffic from different devices to a single individual. That is, determine that similar fingerprints exist for device A and device B, so can determine that the same individual is using both device A and device B. This is where the analysis of network traffic to reverse engineer what services are being accessed is useful. It has been shown that you can fingerprint an individual based on their network activity over a period of time [18]–[22]. We call this pattern a persona. A persona consists of a model of what services are used, when they are used, and for how long. As a simple example that highlights the definition of a person, in Figure 1 we show a collection of web sites that a group of faculty and students use on a daily basis. We can see that each has a different fingerprint. From the same data, we can also see some similarities. As such, while we discuss this in terms of an individual and imply that the interest is in a specific person, a persona can be generalized to be defined as a profile of a class of individuals (e.g., general, medic, faculty, student, executive, software developer, etc.).

Thwarting Persona Tracking with Multiple Personas. In this paper, we propose obfuscating the persona as a solution to an adversary that can track personas as part of the adversary performing pattern of life analysis. This obfuscation may involve generating traffic on a device to look like a desired persona (to mislead the adversary), or generating traffic alongside real traffic to shift the profile of activities to not match a persona (to thwart the adversary's ability to detect a persona).

Regardless of the method, what is needed is to be able to generate traffic in a specific manner according to some desired plan set by the user (the individual trying to avoid tracking, or the organization protecting the individuals). It can be easily understood that both the persona definitions and plans are secret information. If the adversary were able to obtain the plan, for example, they would be able to filter the network traffic to eliminate the fake traffic. In this case, the individual is no longer protected.

As such, in this paper we focus on protecting the software and associated data with secure hardware.

III. Background

This work proposes generating network traffic to impede an adversary's ability to analyze network traffic to perform pattern of life analysis. To ensure secure communication, we leverage existing secure hardware technologies. In this section we provide a background on secure hardware technology, as well as on generating realistic network traffic.

Trusted Execution Environments. The main goal of Trusted Execution Environments (TEEs) is to provide an additional isolation layer beyond the existing separation. As defined by prior work, a TEE is supposed to provide security in four domains: spatial, temporal, communication and fault handling [23]. In the spatial domain, a TEE provides separation of data that resides within the TEE and the data that resides outside the TEE. In other words, data that lives inside the TEE can only be accessed while inside the TEE, while data residing outside the TEE can only be accessed while outside the TEE. On the temporal domain, a TEE provides isolation of resources. A process running inside a TEE cannot leak any information through the shared resources with other processes running either in other TEEs or outside the TEE. On the communication side, the TEE strictly prohibits communication with any component outside of the TEE unless explicitly allowed. Finally, the TEE provides isolation for when a fault

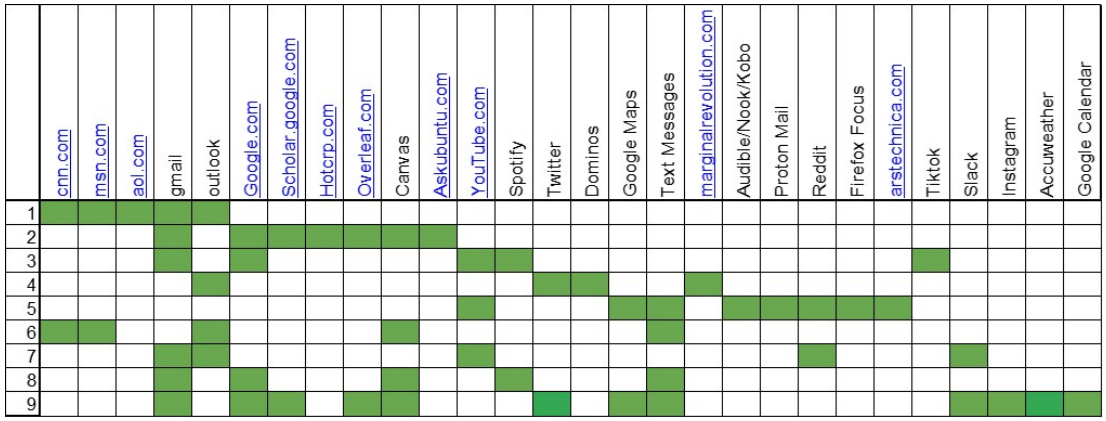


Fig. 1: A sample of a collection of faculty and students showing their on-line activity. The variations, and even similarities illustrate a persona.

is encountered. When a process within a TEE fails, the fault handling cannot affect anything that runs outside of the TEE, and vice-versa.

Secure Memory Confidentiality Some TEEs, such as Intel SGX, provide confidentiality for data in memory [13], [24]. The assumed trusted boundary of Intel SGX is the chip boundary. Any time the processor writes data to the off-chip memory, the data needs to be protected from prying eyes with a cryptographic approach. Whenever the data is fetched from memory it needs to be decrypted before the core can use it. To hide the long latency of decryption and leverage the long latency of a memory fetch, Intel SGX has adopted an encryption mechanism called the AES Galois-Counter method.

Secure Memory Integrity Besides confidentiality, Intel SGX also provides integrity verification for data in memory [24]. Relying solely on counter-mode encryption is not sufficient for protecting the integrity of the data in memory. The threat model for Intel SGX assumes a powerful adversary in physical control of the system. Thus, intercepting the bits going through the bus connecting the memory and processor is possible, enabling splicing and spoofing attacks [25]. Furthermore, as memory technologies continue to shrink, attacks that leverage purposeful disturbance errors in memory are becoming more prevalent [26], [27].

As a result, most state-of-the-art work in secure memory, including Intel SGX, have resorted to computing keyed hash message authentication codes (HMACs) for verifying the integrity of the data in memory [24], [25], [28]. Once again, given that the trusted boundary is the chip boundary, any time the processor accesses off-chip data the on-chip memory controller will verify the integrity of the data fetched. In order to verify the integrity, the on-chip memory controller computes and stores in memory an HMAC whenever a piece of data is updated. When the same piece of data is fetched from memory, the corresponding HMAC is also fetched from memory, recomputed and compared. If the two HMACs are the same, the processor can verify the integrity of the data.

Replay Attack Protection. Hashing data is not enough to protect the system from replay attacks. In a replay attack the adversary replays old values of both the ciphertext and HMAC. Therefore, most state-of-the-art work protects data in memory with an Integrity tree [13], [25], [29].

The state-of-the-art integrity tree (called a Bonsai Merkle tree) is a hash tree of the encryption counters. When a piece of data and its corresponding metadata, are fetched from memory—including the encryption counter and the HMAC—the integrity tree is traversed starting from the leaves of the tree to verify the integrity of the counter. The root of the integrity tree is always on-chip (the trusted boundary) to establish the root of trust while the rest of the tree can be stored in memory. Once the root is recomputed, if the value matches with the on-chip value, then the data fetched from memory can be guaranteed to be free of replay attacks. If the verification fails, the system halts and reboots forcing a cleanup and recomputation of all the data in memory, including its metadata.

Realistic Network Traffic Generation. Network traffic generation and analysis are in purpose-driven. Whether it is generating traffic for capacity and bandwidth tests or analysing traffic for bandwidth estimations and anomaly detection, the traffic is generated with specific characteristics to fulfill a purpose. While real network traffic is characterized by high variability or burstiness and self-similarity, i.e. its distribution remains the same at different time scales [30], traffic generated for a single purpose does not necessarily need to follow the same behavior as long as it fulfills the required criteria. As a result, it can be challenging to generate network traffic that reflects the expected flow with correct packet size distributions, arrival rates, and burstiness [31] as different behavior might raise suspicion under inspection or when trying to detect anomalies.

Various models have attempted to generate realistic traffic by reproducing characteristic variability and self-similarity, including Swing [31], "Sources On Off" [32], or deep learning based models such as NeCSTGen [33]. While these and similar models seek to improve on traditional network traffic generator limitations, they are still subject to limitations such as model training requirements, protocol limitations, or only working in simulated environments. In practice, these models are generally populated from captured packet traces with the

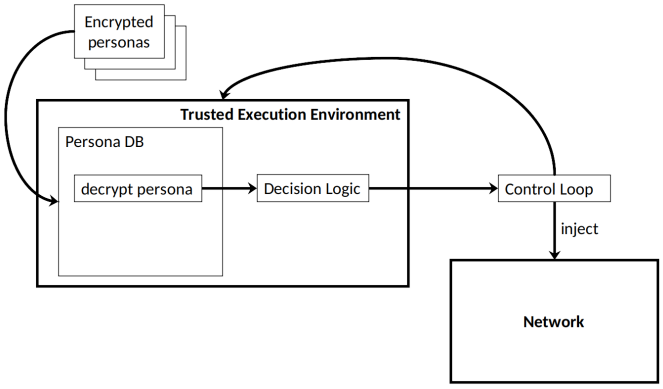


Fig. 2: High-level overview of the proof of concept design.

goal of reproducing the original trace. In fact, while not ideal, the ease, flexibility, and ability to reproduce real behavior of traffic replay tools in traffic generation has been emphasized but discarded due to its inability to generate infinite amounts of traffic and its limitation to simulating individual events [34]. Still, it is exactly this capability of replaying traffic in the form of ”events” that can be used to simulate specific personas and obfuscate fingerprinting.

Obfuscation techniques commonly leverage traffic generation capabilities. Popular methods to obfuscate traffic are padding packets and adding noise to the traffic. However, this makes the assumption that both sides of the communication are participating in the obfuscation. This is challenging in practice, as typical network activity involves visiting websites and using services from third parties. Instead, we rely on replaying interactions with websites and services to generate a persona. This involves a trace of website accesses and making http get requests to each site.

IV. Design

The proof of concept design of the obfuscation framework has three components: the persona database, the decision logic and the control loop. The persona database and the decision logic have to be protected inside the trusted execution environment, while the control loop can reside outside. This design decision arises from the limitations of most TEEs which reduce the communication abilities of the system to minimize the attack vector. The high-level overview of the proof of concept is shown in Figure 2

A. The Persona Database

In order to obfuscate traffic of one person’s activity we first collect a database of network traffic from multiple personas. A persona is defined by the unique network traffic activity as observed by a network administrator. The network traffic activity that can be collected by a network administrator includes the source and destination IP addresses, the content of each packet and the time in between the activity. We use these pieces of information to compose the database of personas.

At initialization time, the network traffic activity from each persona is read from an encrypted file. The file is then decrypted inside the enclave and then read line by line to construct the database of activity for each persona. Each persona’s activity is read from a different file.

B. Decision Logic

Once inside the enclave the persona database is protected by the TEE features from Intel SGX. Similarly the logic used to decide when to inject which persona’s activity is also protected by the TEE. The thread inside the TEE is active at all times and it has one entry function (called an ECALL in the Intel SGX terminology) which can request the next packet’s information.

When a query for a packet is issued, the TEE thread needs to decide which persona to pull from. A naïve approach keeps track of the number of packets replayed from each persona and switches once the maximum number of packets have been reached for each persona. Once all personas have been replayed, the decision logic starts over replaying each persona’s traffic into the network. While this approach is simple, it could reveal the fact that the traffic is not genuine.

A more sophisticated version of the decision logic includes a feedback loop to take into consideration the native network traffic. The goal of the obfuscation platform is to hide any pattern that can be visible to an observer. In order to achieve this goal without revealing that the injected traffic is fake, the decision logic has to maintain a Finite State Machine (FSM) of a pre-loaded action plan. The pre-loaded action plan is based on a mathematical derivation of what needs to happen in order to produce random looking traffic. The finite state machine takes into consideration the native traffic in order to follow the plan accordingly with both fake and native traffic. The FSM drives the logic to either stay in the same persona or switch to the next one.

C. The Main Control Loop

Due to the limitations of TEEs, the main control loop that injects packets into the network needs to reside outside of the TEE. Given that TEE are specifically designed to restrict the communications with the outside world, network interactions are generally not allowed inside a TEE. Thus, at runtime a separate thread outside the TEE queries the secure thread to get the next packet to inject into the network.

The main control loop is a simple function that uses a trusted function (called ECALL in Intel SGX terminology) to query for the next packet to inject into the network. The function returns only basic information such as the source and destination IP addresses, the encrypted packet contents and the presumed duration of the network activity.

Once this basic information is returned to the main control loop, the thread compiles it into a packet and injects it into the network. The duration of the network activity is then used to wait until the next query is issued. This function is active for as long as the framework is running.

V. Results

A. Experimental Methodology

We run the experiments using a Lenovo ThinkCentre M90T running Ubuntu 22.04.02. The machine has an Intel i9-10900

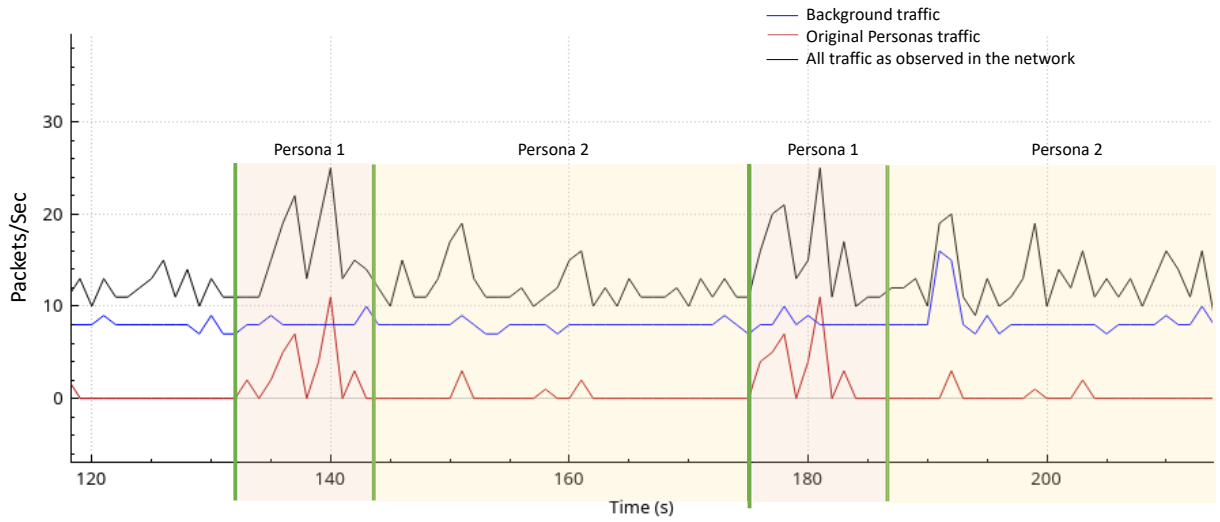
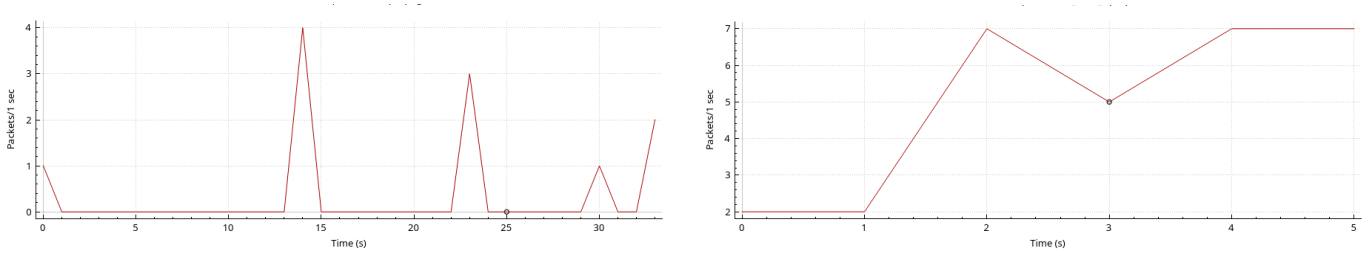


Fig. 3: Network traffic while running two iterations of replicating two personas. The blue line is the background network traffic, the red line is the original personas network traffic combined, and the black line is background and personas network traffic combined.



(a) Persona 1.

(b) Persona 2.

Fig. 4: Individual network traffic for both personas.

CPU running at 2.8 GHz. We let the application run for 10 iterations alternating between two persona definitions per iteration. We run the experiments both inside and outside the enclave to measure the impact of the TEE.

B. Network Traffic Analysis

The main control loop is able to inject traffic packets to simulate two different entities running on the same system. As shown in Figure 3, the control logic is able to inject the packets from each persona seamlessly. The main control logic alternates between the two persona definition leading to a different network traffic pattern. Figure 4 shows the network traffic patterns for each persona individually. By combining both patterns into one device the pattern of life from one user becomes indiscernible.

C. Performance Overhead

Running the logic inside the Intel SGX enclave adds a minimal performance overhead. While Intel SGX overheads are well understood to be large [35], [36], in the case of our proof of concept the overhead is not noticeable. The main reason for this result is the fact that the proof of concept design has a lot of network injection activity which tends to be slower than the expected overhead from TEEs. The biggest impact of the TEE is in the potential for a large burst rate. Intel SGX reduces the burst rate by about 45%. On the other hand, looking at the overall network traffic patterns running with and without SGX, we observe that the packets per second are comparable.

Intel SGX has the biggest impact on performance at initialization. Given that initialization of the persona database does not involve any network traffic injection, the computing needs rely solely on the core and the memory. As a result, the initialization of the proof-of-concept running with SGX incurs a 13.9X overhead compared to running without SGX. This large difference is expected, as the calls in and out of the SGX enclave are costly. While the overhead of using the TEE in this case is extreme, the initialization is only run once at the beginning and should not impact the injected traffic.

VI. Conclusion and Future Work

In this work we have shown that it is possible to hide a user's pattern of life through the network traffic inside a Trusted Execution Environment (TEE). While the TEE incurs some performance overhead during the initialization period,

our experiments show that when it comes to injecting traffic to hide patterns the overhead of interacting with the network surpasses the overhead of interacting with the TEE removing any obstacles from using the technology for obfuscating patterns of life on the network.

REFERENCES

[1] J. Schogol. (2022) Russian troops are proving that cell phones in war zones are a very bad idea. Online; accessed 05-May-2023. [Online]. Available: https://taskandpurpose.com/news/russia-ukraine-cell-phones-track-combat/

[2] A. Coakley. (2021) Borderline: Tinder profiles of polish troops appear in belarus. Online; accessed 05-May-2023. [Online]. Available: https://www.independent.co.uk/news/world/europe/belarus-poland-border-tinder-troops-b1957953.html

[3] A. Priya, S. Nandi, and R. S. Goswami, "An analysis of real-time network traffic for identification of browser and application of user using clustering algorithm," in *2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN)*, 2018, pp. 441–445.

[4] G. Rizothanasis, N. Carlsson, and A. Mahanti, "Identifying user actions from http(s) traffic," in *2016 IEEE 41st Conference on Local Computer Networks (LCN)*, 2016, pp. 555–558.

[5] J. Muehlstein, Y. Zion, M. Bahumi, I. Kirshenboim, R. Dubin, A. Dvir, and O. Pele, "Analyzing https encrypted traffic to identify user's operating system, browser and application," in *2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, 2017, pp. 1–6.

[6] D. F. Maron. (2013) How social media is changing disaster response. Online; accessed 05-May-2023. [Online]. Available: https://www.scientificamerican.com/article/how-social-media-is-changing-disaster-response/

[7] S. A. Diehn. (2013) Revolution evolution. Online; accessed 05-May-2023. [Online]. Available: https://www.dw.com/en/social-media-use-evolving-in-egypt/a-16930251

[8] G. He, X. Xiao, R. Chen, H. Zhu, Z. Zhang, and B. Xu, "Secure and efficient traffic obfuscation for smart home," in *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*, 2022, pp. 6073–6078.

[9] A. J. Pinheiro, P. Freitas de Araujo-Filho, J. de M. Bezerra, and D. R. Campelo, "Adaptive packet padding approach for smart home networks: A tradeoff between privacy and performance," *IEEE Internet of Things Journal*, vol. 8, no. 5, pp. 3930–3938, 2021.

[10] T. Datta, N. Apthorpe, and N. Feamster, "A developer-friendly library for smart home iot privacy-preserving traffic obfuscation," in *Proceedings of the 2018 Workshop on IoT Security and Privacy*, ser. IoT S&P '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 43–48. [Online]. Available: https://doi.org/10.1145/3229565.3229567

[11] E. Ciftcioglu, R. Hardy, K. Chan, L. Scott, D. Oliveira, and G. Verma, "Chaff allocation and performance for network traffic obfuscation," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, 2018, pp. 1565–1568.

[12] J. Brahma and D. Sadhya, "Preserving contextual privacy for smart home iot devices with dynamic traffic shaping," *IEEE Internet of Things Journal*, vol. 9, no. 13, pp. 11 434–11 441, 2022.

[13] I. Anati, F. Mckeen, S. Gueron, H. Haitao, S. Johnson, R. Leslie-Hurd, H. Patil, C. Rozas, and H. Shafi, "Intel software guard extensions (Intel SGX)," in *Tutorial at International Symposium on Computer Architecture (ISCA)*, 2015.

[14] A. Holdings, "Arm security technology: Building a secure system using trustzone technology," 2009.

[15] Y. Wang, Y. Shen, K. Cheng, Y. Yang, C. Su, and A. Faree, "Poster: Obfuscating program control flow with intel sgx," in *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)*, 2018, pp. 321–322.

[16] D. Burihabwa, P. Felber, H. Mercier, and V. Schiavoni, "Sgx-fs: Hardening a file system in user-space with intel sgx," in *2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2018, pp. 67–72.

[17] S. Kim, J. Han, J. Ha, T. Kim, and D. Han, "Sgx-tor: A secure and practical tor anonymity network with sgx enclaves," *IEEE/ACM Transactions on Networking*, vol. 26, no. 5, pp. 2174–2187, 2018.

[18] W. Zhang and H. Wang, "Identification of peer-to-peer traffic based on process fingerprint," in *2011 International Conference on Mechatronic Science, Electric Engineering and Computer (MEC)*. IEEE, 2011, pp. 1559–1562.

[19] Q. Liu, J. Zhang, and B. Zhao, "Traffic classification using compact protocol fingerprint," in *2012 International Conference on Industrial Control and Electronics Engineering*. IEEE, 2012, pp. 147–151.

[20] E. Chan-Tin, T. Kim, and J. Kim, "Website fingerprinting attack mitigation using traffic morphing," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018, pp. 1575–1578.

[21] F. Sun, L. Zhao, B. Zhou, and Y. Wang, "Automatic fingerprint extraction of mobile app users in network traffic," in *2020 5th International Conference on Computational Intelligence and Applications (ICCIA)*. IEEE, 2020, pp. 150–155.

[22] X. Bai, Y. Zhang, and X. Niu, "Traffic identification of tor and web-mix," in *2008 Eighth International Conference on Intelligent Systems Design and Applications*, vol. 1. IEEE, 2008, pp. 548–551.

[23] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted execution environment: what it is, and what it is not," in *2015 IEEE Trustcom/BigDataSE/Ispa*, vol. 1. IEEE, 2015, pp. 57–64.

[24] S. Gueron, "A memory encryption engine suitable for general purpose processors," *Cryptology ePrint Archive*, 2016.

[25] G. E. Suh, D. Clarke, B. Gassend, M. Van Dijk, and S. Devadas, "Aegis: Architecture for tamper-evident and tamper-resistant processing," in *ACM International Conference on Supercomputing 25th Anniversary Volume*, 2003, pp. 357–368.

[26] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of dram disturbance errors," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3, pp. 361–372, 2014.

[27] J. S. Kim, M. Patel, A. G. Yağlıkçı, H. Hassan, R. Azizi, L. Orosa, and O. Mutlu, "Revisiting rowhammer: An experimental analysis of modern dram devices and mitigation techniques," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 638–651.

[28] D. Lie, C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. Mitchell, and M. Horowitz, "Architectural support for copy and tamper resistant software," *Acm Sigplan Notices*, vol. 35, no. 11, pp. 168–177, 2000.

[29] B. Rogers, S. Chhabra, M. Prvulovic, and Y. Solihin, "Using address independent seed encryption and bonsai merkle trees to make secure processors os-and performance-friendly," in *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*. IEEE, 2007, pp. 183–196.

[30] M. Crovella and A. Bestavros, "Self-similarity in world wide web traffic: evidence and possible causes," *IEEE/ACM Trans. Netw.*, vol. 5, pp. 835–846, 1996.

[31] K. V. Vishwanath and A. Vahdat, "Swing: Realistic and responsive network traffic generation," *IEEE/ACM Transactions on Networking*, vol. 17, no. 3, pp. 712–725, 2009.

[32] A. Varet and N. Larrieu, "How to generate realistic network traffic?" in *2014 IEEE 38th Annual Computer Software and Applications Conference*, 2014, pp. 299–304.

[33] F. Meslet-Millet, S. Mouysset, and E. Chaput, "Necstgen: An approach for realistic network traffic generation using deep learning," in *GLOBECOM 2022 - 2022 IEEE Global Communications Conference*, 2022, pp. 3108–3113.

[34] E. Gustafsson, "Synthetic generation of realistic network traffic," 2020. [Online]. Available: http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-165285

[35] T. S. Lehman, A. D. Hilton, and B. C. Lee, "Maps: Understanding metadata access patterns in secure memory," in *2018 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE, 2018, pp. 33–43.

[36] ——, "Poisonivy: Safe speculation for secure memory," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2016, pp. 1–13.