공학박사 학위논문

Information Flow Control for Privacy-preserving Advertising

광고 프라이버시를 위한 정보 흐름 제어 기법

2022년 8월

서울대학교 대학원

컴퓨터공학부

박 민 경

광고 프라이버시를 위한 정보 흐름 제어 기법

Information Flow Control for Privacy-preserving Advertising

지도교수 권 태 경

이 논문을 공학박사 학위논문으로 제출함

2022 년 6 월

서울대학교 대학원

컴퓨터공학부

박 민 경

박 민 경의 공학박사 학위논문을 인준함

2022 년 6 월

위 원 장	김 진 수
부위원장	권 태 경
위 원	백 윤 흥
위 원	전 병 곤
위 원	송 도 경

Abstract

Information Flow Control for Privacy-preserving Advertising

Minkyung Park
Department of Computer Science & Engineering
The Graduate School
Seoul National University

In online advertising, cross-site tracking enables advertisers to target potential customers by profiling their online behaviors. However, such practice raises privacy concerns because of the sensitivity of the collected user data. To address this issue, there have been proposals for privacy-preserving advertising. However, they allow ad companies to protect user privacy only by sacrificing the utility of the advertising. In this paper, I present a privacy-preserving advertising framework, Pave, which allows ad companies to sustain the utility and current advertising mechanisms. Pave provides an arbitrary ad program with an isolated execution environment equipped with a blackbox monitor, called a PaveBox, inspired by Secure Multi-Execution (SME). The PaveBox intercepts every data flow from the ad program and disallows any data flow that may explicitly or implicitly leak

the user data. As the PaveBox is built on top of Intel SGX, its integrity can be remotely attested. I carry out the quantitative analysis with prototype-based experiments to show its feasibility.

Keywords: Information Flow Control, Secure Multi Execution, Intel SGX, Privacy-preserving Advertising, User Privacy

Student Number: 2014-21784

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Online advertising delivers promotional materials (i.e., advertisements) to users over the Internet, which plays a crucial role in the Internet economy. Online users see advertisements by visiting websites, which we call publishers. A publisher shows advertisements in its webpages so that it can get money from the advertisers. The online advertising ecosystem has evolved into a complex marketplace in which various kinds of ad companies try to connect the ad spaces of publishers (e.g., websites and mobile apps) and advertisers. The ecosystem of online advertising typically consists of a few kinds of ad companies who collaborate to determine which advertisement is shown to a user.

To attract a user to click an advertisement and further to increase a chance to purchase the advertised product, ad companies need to identify her profile by using her demographic data or search history. For this, the ad companies keep track of the user behaviors to build up a user profile typically by using third-party tracking techniques. As a user has different third-party cookies for different ad companies, the cookies of the same user may be shared across different ad companies so that they can cooperate to build up a user profile, so-

called cookie syncing. Moreover, Google Advertising ID (GAID) and IDentifier For Advertising (IDFA) for Android or Apple devices, respectively, are used to identify individual devices.

Many believe that targeted advertisements considering user profiles are useful, necessary, or time-saving [18]. On the other hand, many studies [15, 32, 53, 1, 34, 57, 10, 2] warn of the privacy leakage from the targeted advertising due to third-party tracking. While users may be aware of first-party cookies set by the websites they visit directly, they may not be aware of third-party cookies set by third-parties. Google's DoubleClick is reported to be able to collect a user's browsing history of 40% of the top Alexa 100K websites using the third-party cookies in 2015 [15]. Moreover, the ad companies share the user information between one another. Google shares the user's behavior data between its subordinates (e.g., Google Syndication, DoubleClick, and Google Tag Manager) [10]. According to [2], DoubleClick broadcasts personal data to over 2,000 companies, and AppNexus conducts 131 billion personal data broadcasts every day.

Since targeted advertising is one of the main drivers in the Internet economy, targeted advertising while preserving privacy continues to be studied by browser vendors, ad companies, and researchers. Commonly [42, 76, 43, 64, 35], ad servers are given no or only coarse-grained user interests. Hence, they choose a set of candidate ads, and a user agent locally decides an ad from the candidate set.

Such approaches suffer from the coarse granularity of user information, which lowers the utility of the advertisements. Since current ad selection mechanisms target users more specifically, it is difficult for these solutions to be supported. For example, auction-based advertising (i.e., Real-Time Bidding or RTB), which is one of the widely used ad selection mechanisms, heavily relies on choosing advertisements based on highly-targeted users [75]. That

is, it becomes important to narrow targeted users by informing ad servers of fine-grained user data. Thus, the limitation on information granularity may hurt the online ad ecosystem [27]. For example, Google Chrome has delayed multiple times its plan to block third-party cookies until 2023 because of the conflict between the tracking-based advertising and the user privacy [11]. Also, Apple's tracking transparency that blocks tracking user behaviors gives negative impacts on Internet companies (e.g., Meta) that rely on advertising [24]. Furthermore, the existing approaches focus only on a specific mechanism (e.g., ad network [42, 76, 43], ranking-based auction [64], on-device retargeting [35], etc.) and do not explore how to substantiate their designs in the other settings.

I argue that if an arbitrary ad program running on an ad server is monitored by Information Flow Control (IFC), it can process fine-grained user interests to choose an ad while keeping them secure. IFC is a mechanism to enforce a program to follow security requirements by controlling data flows over sensitive channels. Therefore, the ad program is blocked from using the user interests or their derived output after an ad is selected. For example, if the ad program tries to store the user interests into persistent storage, the IFC monitor will block it. Also, I leverage Intel Software Guard eXtension (SGX) to guarantee the integrity and the confidentiality of the IFC monitor. SGX provides an isolated container, called an enclave, which prevents other processes from tampering the IFC monitor. Furthermore, the user agent can remotely attest this enclave to verify the integrity of the IFC monitor.

The key challenge of integrating the IFC monitor with SGX is how to prevent an ad program from leaking user interests through system call (syscall) interfaces. Since privileged software (e.g., operating system) cannot be protected by SGX, implicit leakages through the syscall interfaces should be completely addressed. For example, if the ad program exits only when the user interests include a specific value, whether the `exit` syscall is triggered or not leaks

the value. Unfortunately, existing IFC implementations cannot be directly applied to the SGX settings (see Chapter 9). First of all, they do not address the implicit leakage or incur significant overhead [65, 48, 86, 85, 30, 83, 52, 31]. Moreover, since the program and the IFC monitor have to run with the same privilege (ring 3) within an enclave, it is important to design the IFC in such a way that its enforcement can be implemented. Ryoan [44] is the first work that addresses the challenge by blocking almost all syscalls after reading any protected data. However, this makes it impossible to perform sophisticated ad selections such as RTB since the ad programs require syscalls to interact with each other.

To address this challenge, I present PAVE, a novel combination of techniques for privacy-preserving ad selection. In PAVE, a user agent profiles fine-grained user interests locally, which are sent to an ad server. The user interests on the ad server are protected by 'shadow execution', which is a technique inspired by Secure Multi-Execution (SME) [29, 16]. Like SME, shadow execution simulates the ad program execution without the user interests. The simulated execution thus generates a sequence of syscalls independent of the user interests. Then, the execution with the user interests is safely interleaved into the simulated execution to prevent privacy leakage.

When shadow execution simulates the ad program (without the user interests), an IFC monitor, called a PAVEBOX, replaces the sensitive data by dummy values. Inappropriate dummy values may restrict the functionality of the ad selection program, which may fail to serve an advertisement. Thus, in PAVE, instead of letting the PAVEBOX choose dummy values on its own, they are fed into the PAVEBOX from other entities such as the user agent or the other ad servers. For instance, the user agent sends the context of a website that the user is currently visiting, which is used as dummy values by the ad programs.

Furthermore, PAVE introduces 'faceted session' for the ad programs to exchange the sensitive user data through TLS connections. As the network activities are not protected by SGX, networking-related syscalls would be forbidden by shadow execution. However, in PAVE, faceted session carries both the sensitive data and its dummy values together. Thus, it conceals not only the content of the sensitive data but also its existence.

I make the following contributions. First, I propose a novel privacy-preserving advertising framework, PAVE, which allows ad companies to continue using the current ad selection mechanisms with the fine-grained user data. Second, I design and implement an IFC technique that while an ad program can process user interests, the PAVEBOX prohibits it from leaking the user interests. To the best of our knowledge, this is the first approach to use IFC for privacy-preserving advertising. Third, I prove the security and the functionality of PAVE using a formal model. Lastly, I carry out prototype-based experiments to demonstrate its feasibility.

The rest of the thesis is organized as follows. I present the background of advertising ecosystem, Intel SGX, and Secure Multi-Execution in Chapter 2. Then, I describe the design goals and challenges in Chapter 3. In Chapter 4 and Chapter 5, I detail the design and the implementation of PAVE. Chapter 6 formally proofs the security of PAVE. After that, I discuss the feasibility of PAVE in Chapter 7 and Chapter 8. Finally, the related work is explained in Chapter 9, followed by concluding remarks in Chapter 10.

Chapter 2

Background

2.1 Current Advertising Ecosystem

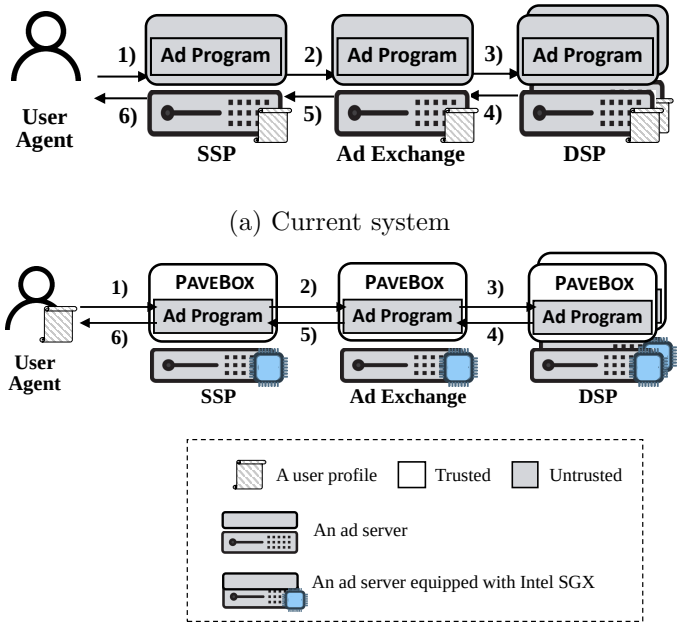
The online advertising ecosystem has become a playground for various kinds of stakeholders. Various ad companies such as an ad network, a supply-side platform (SSP), a demand-side platform (DSP), and an ad exchange (ADX) collaborate in various settings to match advertisers with publishers. There are two widely used ad selection models for coordinating such ad companies: ad network and real-time bidding (RTB). An ad network aggregates ad impression inventories from publishers (i.e., websites that publish ads) and matches them with advertisers. An ad impression refers to an ad view, which is counted whenever an ad is shown to a user. However, as the demand for ad spaces often falls short of the supply, relying on a single ad network is not scalable. Also, the target users are limited to the visitors to the publishers of a particular ad network.

To address such issues, multiple kinds of ad companies decide ads to show in real-time through RTB. That is, special auctions are held so that advertis-

ers can bid to buy impressions. The stakeholders for RTB are ad exchanges, supply-side platforms (SSPs), and demand-side platforms (DSPs). An ad exchange is an online marketplace that enables advertisers and publishers to buy and sell ad spaces. On behalf of the publishers, an SSP sells ad spaces to the marketplace. The SSP allows the publishers to show advertisements to users in such a way that the publishers can optimize their ad spaces automatically. On the other hand, a DSP allows the advertisers to display their ads at the publishers' websites that their target users are visiting now. The DSP purchases advertisements in an automatic fashion by bidding for an impression. Finally, the ads from the highest bidders are displayed on the publishers' websites. Figure 2.1a shows an example of the RTB process.

The ad is delivered as a URL, which is a location to a raw ad material. This is because the advertisement is delivered in the form of an ad markup, which is a representation of an advertisement (e.g., HTML tag). After the ad selection, the user agent loads the raw ad material such as a video or an image as indicated in the URL. Pave focuses on ad selection, which is from a user agent sending an ad request to fetching the corresponding ad URL. For the sake of brevity, 'advertising' refers to the ad selection in this paper. I discuss the remaining processes in Chapter 7. Note that there are various forms of ad selection mechanisms such as the daisy chain and waterfall models. For example, the ad exchanges can buy or sell an ad impression between each other. Or the SSP may send the ad request to ad networks in addition to the ad exchanges. That is, the ad companies collaborate in variable settings for online advertising.

With the explosive growth of the online advertising industry, the ad pricing model has become important to advertisers' budget strategies. Cost per Impression (CPI), Cost per Click (CPC), and Cost per Action (CPA) are common metrics for the pricing models to charge for the impressions of ads. CPI is



(a) Current system

(b) PAVE

Figure 2.1: A process of real-time bidding (RTB) is illustrated.

the rate that an advertiser has agreed to pay per 1,000 views (or impressions) of a particular ad. CPC allows the publishers to charge advertisers for every click on an ad. In the CPA model, publishers charge advertisers only when the user completes the required action. Often, the combination of the above pricing models is used.

2.2 Intel SGX

Intel SGX is a set of instructions added to the Intel processor architecture. It enables an application to build an enclave, a memory area protected from disclosure or modification by any privileged software such as the operating system. Some area of the main memory is reserved for enclaves, which is called Enclave Page Cache (EPC). The EPC is encrypted by the Memory Encryption

Engine (MEE) and decrypted only when it is loaded to the CPU. Intel SGX also provides the remote attestation that allows an entity (say, challenger) to verify the integrity of the code and initial data of an enclave in a remote server. That is, the challenger checks the signed hash of the initial memory of the enclave.

One of the threat models of Intel SGX is the privileged software outside the enclave. Therefore, operations that need the intervention of the privileged software are prohibited. For example, Intel SGX blocks an enclave application from directly invoking syscalls to access the system resources (e.g., file, time, or network). Instead, an enclave developer can define an OCALL interface to call a host function outside the enclave. A thread temporarily exits the enclave, executes a host function, and re-enters the enclave.

2.3 Secure Multi-Execution (SME)

Non-interference is a security property that IFC seeks to satisfy. A program is said to be non-interferent if its non-sensitive public outputs (i.e., outputs at a low security level) cannot be influenced by sensitive inputs (or inputs at a high security level). Especially, it is *timing-sensitive* non-interferent when the program satisfies non-interference at any time of its execution. This means that sensitive inputs cannot be leaked to timing channels as well. For the sake of brevity, 'non-interference' refers to timing-sensitive non-interference in this thesis.

Suppose that an ad server runs a DSP program below.

```
1  user_interest = read_sensitive();
2  bid = bidding_function(user_interest);
3  send_to_TTP(bid);
4  save_to_shared(bid);
```

The program takes the user interest as sensitive input (line 1). The pro-

gram determines the bid price by processing the user interest (line 2); thus, the output (i.e., `bid`) becomes sensitive. The bid price is then sent to a trusted third-party (line 3). In this example, I assume that a trusted third-party plays the role of the ADX. In other words, I consider `send_to_TTP` as a secure output channel. Lastly, the program saves the user interest to a shared storage (line 4). I assume that the shared storage is accessible by anyone and consider `save_to_shared` as an insecure output channel. Consequently, this DSP program does not satisfy non-interference because the sensitive output (`bid`) is transmitted over the insecure channel (`save_to_shared`).

Secure Multi-Execution (SME) [16, 29, 45] enforces non-interference on any given program. The key idea is to execute a given program multiple times, once for each security level by the following two principles. First, an execution is disallowed to read input from a channel of a higher security level. Second, an execution is only allowed to output data to the channel of the same security level.

For the above DSP program, there are two security levels: low and high. Thus, SME will spawn two executions, which I refer to as a *low execution* and a *high execution*. In the low execution, `user_interest` is replaced with a predefined dummy value, say 0, since it is a sensitive (i.e., high security level) input. Also, `send_to_TTP()` is ignored as it is a secure output channel.

```
1  user_interest = ~~read_sensitive()~~ 0;
2  bid = bidding_function(user_interest);
3  ~~send_to_TTP(bid)~~;
4  save_to_shared(bid);
```

Meanwhile, during the high execution, `save_to_shared()` is ignored because it is an insecure output channel. This way, SME ensures that the sensitive `user_interest` value is not leaked to the shared storage.

```
1  user_interest = read_sensitive();
```

```
2  bid = bidding_function(user_interest);
3  send_to_TTP(bid);
4  save_to_shared(bid);
```

Chapter 3

Overview

Figure 2.1b illustrates how an ad is selected in PAVE. Compared to Figure 2.1a, the entity that collects user interests is changed from ad servers to a user agent. Unlike the current advertising, the ad servers do not need to track user browsing histories. The user agent sends an ad request along with the user interests. Every ad server processes the ad request under the eyes of a PAVEBOX, which is protected by Intel SGX in turn. Eventually, the user agent receives the corresponding ad URL as in the current practice.

3.1 Threat Model

I assume that a user agent is benign. Usually, browser vendors try to protect user privacy [82, 88] and it is difficult to compromise a user agent. Moreover, any malicious behaviors to compromise the user agent may be detected by various solutions such as anti-virus software. The user agent having the user interests does not trust the ad companies to keep them secret.

When an ad program is protected by an enclave, a host outside the enclave

cannot directly observe the ad program's execution. However, a malicious ad company may develop its ad program in such a way that the user interests are revealed when the ad program has to escape the enclave to access host resources such as time, network, and so on. In this thesis, I focus on the scenario in which the adversary exploits *software interfaces* [44, 4] between an enclave and a host. The software interfaces are implemented as OCALLs, and thus the adversary outside the enclave can observe their invocations and parameters (as mentioned in §2.2). Note that not every syscall needs to be implemented as the software interfaces. For example, `getrandom()` can be implemented using Intel SGX RDRAND instruction. Or, system features can be integrated using LibOS such as Graphene-SGX [77]. For the sake of brevity, I use 'syscall' to refer to syscall implemented as the software interface.

As to the related work on SGX-based IFC [44, 4], I consider micro-architectural vulnerabilities [79, 20, 80] and side-channel attacks [67, 61] out of scope. I discuss the impact of these attacks and countermeasures on PAVE in Chapter 7. Also, denial-of-service attacks are out of the scope of our threat model.

3.2 System Goals

The main goal of PAVE is to achieve (timing-sensitive) non-interference of an ad program. It means that patterns (i.e., order, timing, and passed parameters) of the software interfaces should not leak the user interests. For this, the PAVEBOX controls an ad program's behaviors to ensure that the program execution exhibits a consistent pattern irrespective of the values of user interests.

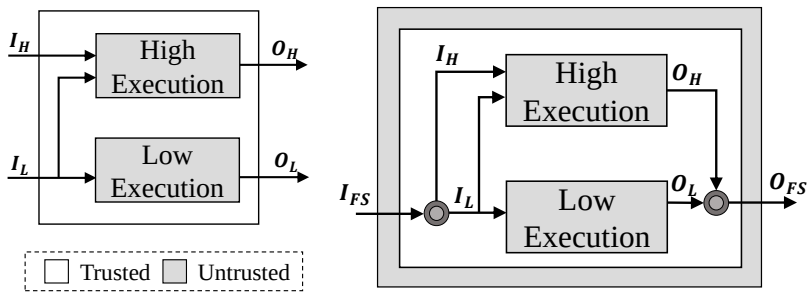
In addition, our system seeks to preserve the functionality of an ad program. Excessive restrictions on the ad program may hinder its important functionalities. For example, if the non-interference is achieved by forbidding every syscall, the ad program cannot return an ad URL. Thus, the PAVEBOX should not restrict or change the functionality of the program if it does not harm non-

interference. Specifically, if an ad program is non-interferent without PAVE, its execution should make the same result in PAVE.

To achieve the goals, I propose 'shadow execution', which is a technique inspired by SME. To be brief, shadow execution runs the ad program twice: a *low execution* with a dummy profile and a *high execution* with a real user profile. The low execution makes syscall invocations, which are independent of the real user interests. By contrast, the high execution cannot invoke syscalls. Instead, it reuses the syscall results from the low execution. That is, the high execution is "safely" interleaved with the low execution. In this way, the high execution is prevented from leaking the user interests through syscalls.

However, there still remain three challenges. The first challenge is how to allow a safe exchange of sensitive data (i.e., the user interests and its derived outputs) over a network. The network is an insecure channel since every operation to use the network is triggered by a syscall. As illustrated in Figure 3.1a, SME assumes that the high execution has a dedicated secure channel over which it can safely read or write sensitive data (I_H or O_H). However, in PAVE, there cannot be such a channel due to the intervention of a software interface. In the example in §2.3, `read_sensitive` is labeled high. The low execution will be fed the dummy value by the SME runtime monitor. The high execution is supposed to get the actual user interests, which is not possible due to the software interface. Thus, I should devise a method to securely transfer sensitive data over the insecure channel.

I propose *faceted session* to address this issue. the PAVEBOX retains two logical sessions over one TLS session. Outside the enclave, the session looks like a single encrypted session between the user agent and the ad program. The endpoint that can encrypt or decrypt the TLS session is not the ad program but the PAVEBOX. When the low execution invokes syscalls to read any sensitive data, the PAVEBOX copies the data from the host to the enclave, but



(a) SME (b) PAVE

Figure 3.1: SME and shadow execution of PAVE are compared.

gives dummy values to the low execution. On the other hand, when the high execution invokes syscalls to read any sensitive data, the PAVEBOX returns the real data, which were obtained while performing the low execution. As illustrated in Figure 3.1b, every sensitive data is delivered through the low channel (I_{FS} or O_{FS}) but the sensitive data is hidden (I_H or O_H).

The second challenge is how to determine dummy values. When the low execution tries to read sensitive data, the PAVEBOX will replace them by dummy values. As the syscall invocation is fully delegated to the low execution, the dummy value can affect the result of the syscall invocation. Thus, if the dummy value goes beyond original semantics (e.g., a garbage value for a user interest), the functionality of the high execution may become ineffective. Also, since the ad programs communicate with each other, any derived output (e.g., bid) from the user interest requires its dummy value as well. In PAVE, the dummy value is not predefined in advance. Instead, the user agent generates a dummy value from the user's current context, which is propagated along with the ad servers. It helps the low execution make syscalls relevant to the ad selection.

The last challenge is how to schedule both high and low executions inside the enclave. SME preserves the functionality under a low-priority sched-

uler [29]. To schedule both executions, a scheduler should be able to learn the states of the executions, but a host scheduler cannot learn the states protected by SGX. Also, it is hard to implement the low-priority scheduler inside the enclave because enclave threads are running with the same Ring 3 privilege. For this, I propose synchronization in PAVE. Both of the executions are scheduled by the synchronization inside the enclave without relying on the host scheduler or the low-priority scheduler.

Chapter 4

Pave Design

4.1 Local Profiling

Pave requires the user agent to profile its user. The user agent keeps its local database of the user interests. Each webpage can declare a context profile, which is a set of interests (e.g., swim, classic music, wine, etc.) representing the page, by using an HTML meta tag. For example, a shopping page for running shoes can declare `sport`, `running`, and `shoes`. Or, for retargeting, its product ID can be declared. Then, when the user visits the webpage, the user agent updates the local database. Each interest is an element from the universal profile set (say, [72]) containing all the possible interests and publicly available. I assume that user interests can include demographics data such as language, age group, and so on. The universal profile set should not contain controversial or highly-sensitive interests for advertising such as race, sexuality, etc.

When the user agent visits the publisher website, it sends the user interests (or the user profile) to an ad program. The user profile is a partial set of

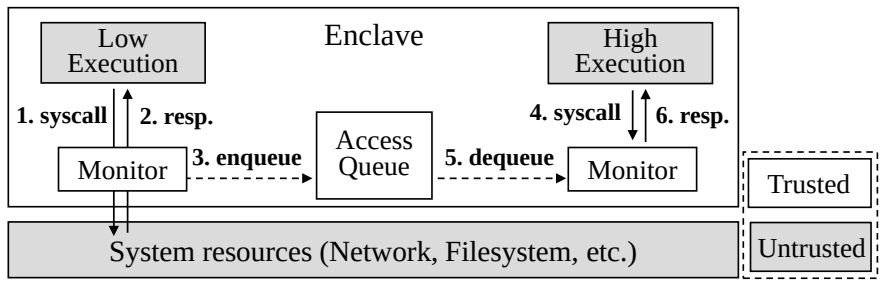


Figure 4.1: Syscalls are controlled differently depending on whether the execution is high or low.

interests from the local database and is determined by the user agent. I do not constrain how a user profile is determined. For example, it can select randomly or frequently appeared interests from the local database. In addition to the user profile, the user agent also sends the context profile of the currently visited webpage. This context profile would be used as dummy values. Such contextual advertising is not privacy-invasive [23] since this website already is known to ad servers when an ad request is triggered. For example, the HTTP referrer header identifies the address of the webpage.

Before sending the pair of profiles, the user agent should remotely attest the ad program to verify whether it is monitored by a PaveBox. If the attestation fails, the user agent does not proceed further. The remote attestation is detailed in §4.4.

4.2 Shadow Execution

When an ad program processes the ad request from the user agent, the PaveBox should protect the user profile (not the context profile) from the leakage through syscalls (i.e., the software interfaces). To achieve this, the PaveBox simulates every invocation in the low execution with the context profile. The PaveBox gives this result to the high execution that runs with the user profile.

Given an ad program, the PaveBox executes its two copies running in parallel. The PaveBox maintains a request-response syscall queue, called an access queue, which is illustrated in Figure 4.1. When the low execution issues a syscall, the PaveBox hands it over to the OS (out of the enclave). Then, the PaveBox delivers the corresponding response back to the low execution. The pair of the issued syscall and the corresponding response is also stored in the access queue. On the other hand, a syscall from the high execution is handled directly from the access queue. The PaveBox dequeues the entry of the same request from the access queue, and its response is delivered to the high execution. If there is no corresponding request, the high execution waits until the request is triggered from the low execution and its response arrives at the queue.

Note that the termination is also handled by the low execution. If the low execution terminates earlier, both executions are terminated. If the high execution terminates before the low execution does, the PaveBox holds its termination and makes the high execution wait until the low execution is terminated. That is, the low and high executions always terminate at the same time.

The two executions should be separated from each other to block access to data of the other counterpart. Likewise, the PaveBox should be protected from the two executions. For this, I use Software Fault Isolation (SFI) [74], which is detailed in Chapter 5.

4.3 Faceted Session

Ad programs should interact with one another (say, between an ADX and a DSP) through the network, which is an insecure channel due to its observability by an adversary. A faceted session enables the high execution to deliver a message with sensitive data over the network.

A faceted session (for short, a session) refers to a network channel between (i) the user agent and an ad program, or (ii) two ad programs. A session is encrypted by the PaveBox (not the executions) through TLS so that the adversary cannot read the content of messages. Also, the PaveBox pads every message to a fixed size to hinder the traffic analysis. In addition, PaveBoxes should be attested when a session is set up.

Every session message should follow a predefined data format, called an AD Message (ADM), whose structure is similar to key-value pairs. Each entry in an ADM has three components: key, value, and security level. The key is an index of the entry, and a value contains the data of the entry. The security level indicates whether the entry is sensitive (`S`) or non-sensitive (`NS`). The value is composed of a pair of raw data, one for the low execution (`L`) and the other for the high execution (`H`). I call this composite data as a *faceted value*. The below illustrates an ADM for an initial ad request where `age`, `sport`, and `interest type` entries are `S`, while `ad placement` one is `NS`.

```
1 "ad placement", NS, {L:"top", H:"top"}
2 "interest type", S, {L:"sport", H:"age, sport"}
3 "age", S, {L:null, H:"30s"}
4 "sport", S, {L:"football", H:"tennis"}
```

To send an ad request, the user agent constructs an ADM from the user profile and the context profile. For the `S` entries, it fills its `H` field with the user profile and `L` field with the context profile. For the `NS` entries, it fills both `H` and `L` fields with the same data.

When an execution accesses (reads or writes) an entry, the PaveBox enforces it to access either `H` or `L` component depending on its level. Every ADM is managed by the PaveBox; that is, the execution cannot directly access it. Instead, the PaveBox passes/updates the corresponding field in the ADM to the execution (when reading), or from the execution (when writing). Conse-

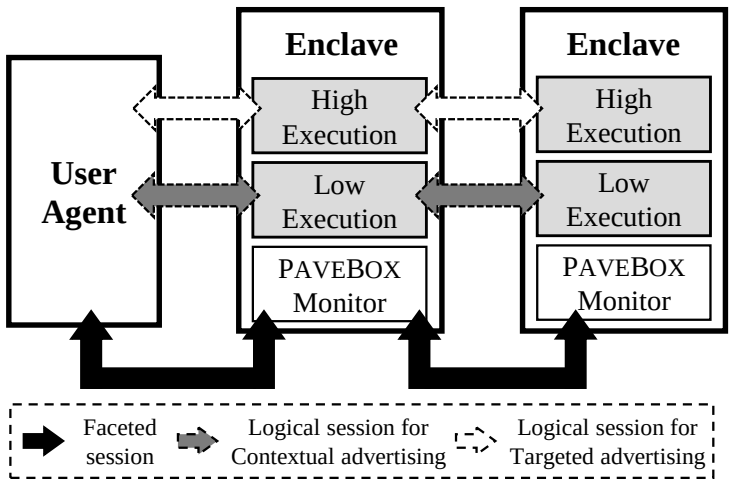


Figure 4.2: TLS carries data for the faceted session so that PAVE performs (i) targeted advertising among high executions and (ii) contextual advertising among low executions.

quently, the high executions in the distributed ad servers process the real user profile, while the low executions process the context profile as illustrated in Figure 4.2.

The transmission or reception of an ADM triggers a syscall. Thus, the low execution decides when to send or receive an ADM. If the low execution sends an ADM before the high execution finishes the `H` field of an entry, the recipient of the ADM may not obtain the data needed for a proper ad selection. Our synchronization mechanism prevents such a situation as detailed in §5.3.

Lastly, the ad server connected to the user agent will return an ad URL. That is, the user agent will receive an ADM with two ad URLs; one in the `H` field is tailored to the user profile, and the other in the `L` field is selected based on the context profile. On receipt of the ad response, the user agent reads the ad URL in the `H` field.

Unfortunately, the user agent may receive an ADM with the URL in the `H`

field missing. For example, if an SSP program is interferent, its high execution may be stuck while waiting for a syscall result. Thus, if the user agent receives an empty ad URL in the `H` field, it uses the ad URL in the `L` field. Even though this ad is of less utility, it can be still meaningful to the user.

4.4 Chained Attestation

The user profile should be delivered to the next entity only if it is attested via faceted session. Since the user agent connects to only one PaveBox (usually, in an SSP), other PaveBoxes behind it are invisible and unknown. Thus, after the user agent attests the directly connected PaveBox, the next hop PaveBoxes are attested by the attested PaveBox. For this, when initiating a session, a client-side PaveBox remotely attests a server-side PaveBox. For instance, in RTB, an SSP is attested by the user agent, an ADX is attested by the SSP, and a DSP is attested by the ADX.

Furthermore, to prevent a user profile from leaking to another user agent, a pair of high and low executions should process a single ad request at a time. After an execution pair accepts an incoming session (from the user agent or another execution pair), it cannot accept another session. It does not mean that the PaveBox cannot handle multi-threading. The PaveBox can run multiple pairs of executions, each of which handles a single ad request at a time. Note that there is no restriction on the number of outgoing sessions.

These rules make execution pairs form a tree topology where nodes are the execution pairs and edges are their sessions. As a root, the user agent attests the SSP PaveBox, the other nodes (i.e., PaveBoxes) are iteratively attested along with the tree links by the remote attestation. Consequently, only executions monitored by PaveBoxes can participate in the ad selection in Pave. Note that the client attestation is not required so that the user agent does not need to be attested.

Chapter 5

Pave Implementation

In this chapter, I detail the implementation of PaveBox to enforce the Pave IFC rules.

5.1 Sandbox

As described in Chapter 4, I use SFI [74] to segregate one enclave into spaces for two executions and a PaveBox monitor. Since one enclave is a single process domain, the high and low executions without SFI can access the memory space of their counterparts (e.g., the low execution reading `H` fields). For SFI, Pave uses address masking by extending Google Native Client (NaCl) [68]. NaCl enforces every memory access of an untrusted program (i.e., an ad program) to point to a designated memory region. It is an aligned 4GB region, and its base address is stored in a designated register (i.e., `r15`), which is forbidden to be modified. Then, an instruction of address access is masked by using `r15` to be in the range [`r15`, `r15+4GB`). In the example below, the first instruction makes `r11` under 4GB and the second instruction masks it using

r15. Consequently, rip after jmp will point to one of the address range [r15, r15+4GB).

```
and 0xfffffffe0, %r11d # upper 32-bits are cleared
add %r15, %r11
jmp *%r11
```

In PAVE, the memory regions for code and data are distinct. While masking any rip-modifying instructions (e.g., jmp) from NaCl is preserved, memory access for data is changed to point to a data region assigned to each execution. Figure 5.1 illustrates that individual executions share the code area, and each has its private data area. As the base address of the data region is initialized to r14, each execution is forced to access its own data region by address masking using r14. The example below shows that rbx is masked to store the data in rax.

```
mov %ebx, %ebx # upper 32-bits are cleared
add %r14, %rbx
mov %rax, (%rbx)
```

Specifically, I extend NaCl by the following rules.

- The memory space for an execution consists of a shared code region and a private data region. Each region is 4GB-aligned and unwrapped.
- The base address of the data region is initialized to r14, which is forbidden to be modified.
- All instructions of indirect data access should use r14 for its address to be in the range [r14, r14+4GB).
- rsp and rbp can be modified by copying each other without masking. For other cases, rsb and rbp should also be masked using r14.
- Any instruction using a rip-relative address is forbidden.

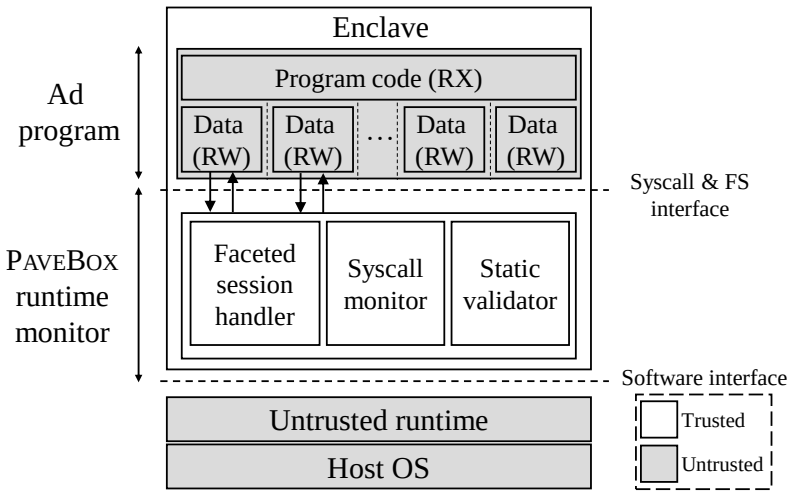


Figure 5.1: A single instance of the PaveBox is illustrated.

Since the address masking for multi-domain has been studied in the literature [4, 13, 69], I skip its technical details. Note that an ad program is loaded into designated address regions, which is used for remote attestation (see §5.4).

I provide a toolchain to instrument the address masking. Given C/C++ source codes, it generates an address-masked executable. Also, to check any violation, as soon as the PaveBox starts, the PaveBox validator statically analyzes whether every memory access is correctly masked or not. If the validation fails, the PaveBox aborts the program.

5.2 Syscall and FS Interfaces

For an execution to access a faceted session (or ADM), I introduce faceted session (FS) interfaces as follows.

- `accept()` accepts an incoming session. It returns a session identifier `sid`.
- `connect(IP, port)` connects to a remote PaveBox using its IP address and port number. It returns a session identifier `sid`.

- `create_adm()` creates an empty ADM and returns its identifier `admfd`.
- `send_adm(sid, admfd)` sends the `admfd` through the session `sid`.
- `receive_adm(sid)` receives an ADM from the session `sid` and returns an ADM identifier `admfd`.
- `get_adm_entry(admfd, key, security)` returns the value of the matched entry in the `admfd` using a given `key` and the `security` level of the entry.
- `add_adm_entry(admfd, key, value, security)` adds an entry to the `admfd`. It is populated with the `key`, `security` and `value`.

Each interface is implemented by using a trampoline, which is a piece of code to transfer control from an execution to the PaveBox monitor. The trampoline code is loaded in the untrusted area into which the execution can jump. Here, the trampoline code is attested by the remote attestation so that the trampoline can contain sensitive instructions to jump to a PaveBox monitor. After the control is transferred to the PaveBox monitor, the monitor handles the interface (e.g., copying an ADM to the execution memory) and the control is returned back to the execution. In NaCl, the trampoline is used to implement syscalls, while in Pave it enables the PaveBox to intercept every syscall. Figure 5.1 illustrates that the syscall and FS interfaces are between the untrusted ad program and the trusted monitor. I provide libc and the library for the FS interfaces for developers to use easily.

5.3 Synchronization

Synchronization of the high and low executions is important to preserve the functionality of ad programs. For example, given a non-interferent ad program, if the low execution proceeds faster than the high execution, the high execution

must be terminated by the low execution even though it has not yet completed its job.

Thus, the PaveBox explicitly synchronizes the two executions. It schedules the two executions in a unit of a basic block to make them keep pace with each other in terms of the number of executed basic blocks. For this, the Pave toolchain instruments the program by inserting a sequence of synchronization instructions to every basic block. The inserted instructions transfer the control of the running thread to the PaveBox scheduler. The scheduler will check how many basic blocks were executed by the low and high execution respectively. This way, it figures out which thread should be suspended. In addition, the toolchain should insert the synchronization instructions before every `send_adm()` call, too. This is to ensure that the faceted ADM to send is properly filled in by both of the low and high executions before the transmission.

Note that the PaveBox validator does not check whether the program is properly instrumented with the synchronization instructions. This is because the non-interference property is guaranteed even without the synchronization. I formally discuss this in Chapter 6.

5.4 Selective Remote Attestation (RA)

The remote attestation is a process to allow an entity (or a challenger) to verify the integrity of the initial memory of an enclave in a remote machine. The challenger checks MRENCLAVE, which is a hash of the initial memory. It is signed by the quoting enclave (in the remote machine), which is a special enclave to perform the remote attestation, and the signed data structure is called a quote. When the enclave (to be attested) sends its quote to the challenger, the challenger asks the Intel Attestation Service (IAS) to verify the quote. Then, the IAS returns an attestation verification report, which tells

whether the quote is forged or not.

The MRENCLAVE is a hash of a buffer, which is extended by ECREATE, EADD, EEXTEND, and EINIT instructions on Intel SGX. ECREATE initializes the buffer with the size of an enclave. When EADD loads a page to the EPC, the buffer is extended with the metadata of the loaded page (e.g., RWX permission, size, page type, and offset). Likewise, EEXTEND extends the buffer with the content of the loaded page. Lastly, EINIT finalizes the MRENCLAVE by hashing the buffer and the MRENCLAVE cannot be changed since then. Consequently, EADD and EEXTEND are especially related to a loaded binary.

Since an ad program is not open to the public, the challenger needs to attest the PaveBox runtime without knowing the ad program. Thus, the Pave loader selectively measures the enclave by using that the ad program is loaded into the designated address regions (as explained in Chapter 5). The Pave loader measures the metadata and content of the PaveBox runtime through EEXTEND, but it only measures the metadata of the ad program. Since EADD measures the metadata (of each page) of the ad program when it is loaded, the challenger can prove that the ad program is loaded into the designated memory area. Even though the content of the ad program is not measured, it will be validated by the static analyzer as soon as the PaveBox starts. I apply this selective RA to RA-TLS [46], which is a TLS extension to do the remote attestation during the TLS handshake.

5.5 Lazy Execution

Shadow execution uses almost twice as many thread resources as standard execution. To reduce the redundancy, the PaveBox delays shadow execution as much as possible. In the beginning, there is only one thread running. Once an entry in the received ADM is read, a new thread is assigned to the high exe-

cution; the original thread becomes the low execution. To reduce the runtime overhead for creating a new thread, the PaveBox creates one in advance.

Chapter 6

Formal Analysis

In this chapter, I present the formal model of a PaveBox and discuss its security properties. Specifically, our model (i) enforces non-interference on any program; and (ii) preserves the functionality of a secure program.

6.1 Model Language

I start by presenting the syntax of our model language in Figure 6.1. A syscall command `recv` reads an ADM, whereas a `send` command transmits an ADM to an external host. A `get` command is used to read the value for key k from an ADM. Similarly, an `add` command is used to update the value for key k. I assume that the sets of sensitive and non-sensitive entry keys are predefined as K_S and K_{NS}, respectively. For example, a `get` command with $k \in K_S$ means an access to a sensitive entry.

For conciseness, I omit several features that are not directly related to the key idea of our proof for security properties. First, I assume that there is only one session which is already established, and a program processes only one

```
Command c  ::=  recv
            |   send
            |   get k to x
            |   add x to k
            |   skip
            |   x := e
            |   c; c
```

Figure 6.1: The syntax of our model language is presented.

ADM at a time. Also, I omit the syntax for conditionals/loops and syscalls that are unrelated to the ADM. These can be handled in the same way as in the formal model of SME [29]. While it is straightforward to extend our model to support such features, I will focus on the difference between the original SME [29] and a PaveBox.

6.2 Standard Semantics

Next, I present the standard semantics of our language. I define the semantics as transition rules of an execution state $\langle c, m, a, I, O \rangle$. Here, c represents the remaining commands to execute. Memory m is a mapping from a variable to a value. Intermediate map a temporarily contains an ADM after it is received, or before it is sent out. Input I denotes the list of ADM to receive, and output O is the accumulated list of the transmitted ADM.

Figure 6.2 presents relation $\rightarrow$ that describes the transition rules of an execution state. The standard semantics describes the original behavior of the program expected by developers who are unaware of PaveBox. Therefore, the standard semantics does not have the concept of a faceted value discussed in §4.3. Here, $eval(e, m)$ represents a function that evaluates the given expression e under the provided memory state m. Also, I use $m[k \mapsto v]$ to denote a map

$$(1) \quad \frac{I = [a_1, a_2, ..., a_n] \quad I' = [a_2, ..., a_n]}{\langle \texttt{recv}, m, a, I, O\rangle \rightarrow \langle \texttt{skip}, m, a_1, I', O\rangle}$$

$$(2) \quad \frac{O = [a_1, ..., a_n] \quad O' = [a_1, ..., a_n, a]}{\langle \texttt{send}, m, a, I, O\rangle \rightarrow \langle \texttt{skip}, m, a, I, O'\rangle}$$

$$(3) \quad \frac{a(k) = v \quad m' = m[x \mapsto v]}{\langle \texttt{get } k \texttt{ to } x, m, a, I, O\rangle \rightarrow \langle \texttt{skip}, m', a, I, O\rangle}$$

$$(4) \quad \frac{m(x) = v \quad a' = a[k \mapsto v]}{\langle \texttt{add } x \texttt{ to } k, m, a, I, O\rangle \rightarrow \langle \texttt{skip}, m, a', I, O\rangle}$$

$$(5) \quad \frac{eval(e, m) = v \quad m' = m[x \mapsto v]}{\langle x := e, m, a, I, O\rangle \rightarrow \langle \texttt{skip}, m', a, I, O\rangle}$$

$$(6) \quad \frac{}{\langle \texttt{skip}; c_2, m, M, I, O\rangle \rightarrow \langle c_2, m, M, I, O\rangle}$$

$$(7) \quad \frac{\langle c_1, m, a, I, O\rangle \rightarrow \langle c_1', m', a', I', O'\rangle}{\langle c_1; c_2, m, a, I, O\rangle \rightarrow \langle c_1'; c_2, m', a', I', O'\rangle}$$

Figure 6.2: Standard execution semantics is defined.

obtained by updating m to have value v for key k. In addition, I consider $m(k)$ to return a bottom (e.g., $\perp$) if key k is not found in map m.

6.3 PaveBox Semantics

Now I present the semantics of program executions under a PaveBox. As I discussed in §4.2, PaveBox separates a program execution into a low execution and a high execution to ensure non-interference. In Figure 6.3, rules (L1) through (L11) define relation $\Rightarrow$ that corresponds to the *local* semantics for a low or high execution. Local state $\langle c, m\rangle_l$ denotes remaining commands c and memory m for the execution level l, which can be either low (L) or high (H). Other components such as a, I, and O are shared by the two execu-

tions. Meanwhile, rule (G1) defines relation $\rightsquigarrow$, which is the global execution semantics over both low and high executions. As described in §5.3, this rule states that the low and high executions run in a synchronized manner, with an alternating order.

Note that syscall operations are fully delegated to the low execution, as described in (L1) through (L4). The semantics of syscall operations ensure that an ADM is maintained in a faceted form within PaveBox, which I elaborate shortly after. Rules from (L5) to (L7) show that an access to a faceted ADM operates on different components of the faceted value, depending on the execution level.

To define the low execution semantics for `recv`, I introduce a function that spawns a faceted ADM. For an ADM entry key k and value v, I define a pre-processing function $fct(k, v)$ as $\langle v, v \rangle$ if $k \in K_{NS}$ and $\langle v_{def}(k), v \rangle$ if $k \in K_S$, where $v_{def}(k)$ is the dummy value for k. I can also extend this for an ADM a, by defining $fct(a)$ as $\{(k, v') \mid (k, v) \in a,\ v' = fct(k, v)\}$. While this operation actually takes place in coordination with the user agent (see §4.3), I abstract this process and embed it in the semantics of PaveBox.

Next, for `send` syscall, the low execution should construct an output ADM by extracting proper components from the faceted ADM. I define a post-processing function $ext(k, \langle v_L, v_H \rangle)$ as v_L if $k \in K_{NS}$ and v_H if $k \in K_S$. Again, I extend it over a faceted ADM a, by defining $ext(a)$ as $\{(k, v') \mid (k, v) \in a,\ v' = ext(k, v),\ v' \neq \bot\}$. Similarly to the pre-processing step, this operation actually occurs in coordination with the user agent, but I simplify this and include it in the PaveBox semantics.

Another notable point is that (L11) ensures that the high execution does not halt even after the remaining command is reduced to a `skip`. In contrast, the low execution halts in such a condition. The global execution will finish only when the low execution halts.

Recall from §5.3 that PaveBox does not validate whether a program is properly instrumented for the synchronization. Thus, an adversary may remove the synchronization instructions from an instrumented program. Figure 6.4 defines the global execution semantics without an assumption of the synchronization. Here, the global state contains a counter n, which is initialized to 0. Relation $\overset{f}{\dashrightarrow}$ is defined with respect to a function $f : \mathbb{Z} \to \{\mathtt{L}, \mathtt{H}\}$ that specifies the order of the executions. For instance, if I use $f_{sync}(n) = (L \texttt{ if } n \texttt{ mod } 2 = 0,\ H \texttt{ otherwise})$ as f here, $\overset{f}{\dashrightarrow}$ gets equivalent to $\rightsquigarrow$ in rule (G1) of Figure 6.3. As I formally state in §6.4, PaveBox can enforce the non-interference under $\overset{f}{\dashrightarrow}$ for any function f.

6.4 Security Properties

First, I extend the notations for semantic rules. For any transition $\hookrightarrow$, I define $\hookrightarrow^n$ to mean a transition performed by applying $\hookrightarrow$ for n times. Also, I abuse $\hookrightarrow^n$ by defining it over the essential input and output components. For example, with standard semantics on program P and input I, if $\langle P, m_0, a_0, I, O_0\rangle \to^n \langle c, m, a, I', O'\rangle$, I will also say that $(P, I) \to^n (I', O')$ holds. Here, m_0 and a_0 are empty maps, and O_0 is an empty list.

Next, I introduce the concept of equivalence between two inputs (or two outputs). For non-faceted ADMs a and a', I say $a{=_{NS}}a'$ iff $dom(a) = dom(a')$ and $\forall k \in dom(a) \cap K_{NS},\ a(k) = a'(k)$. Similarly I say $a{=_S}a'$ iff $dom(a) = dom(a')$ and $\forall k \in dom(a) \cap K_S,\ a(k) = a'(k)$. Now, I extend $=_{NS}$ on ADM lists $A = [a_1, ..., a_n]$ and $A' = [a'_1, ..., a'_m]$. I say $A{=_{NS}}A'$ iff $len(A) = len(A')$ and $1 \leq \forall i \leq n,\ a_i{=_{NS}}a'_i$, where len returns the length of the list. I can extend $=_S$ in the same way.

Definition 1 (**Non-interference**). Assume a program P, semantics $\hookrightarrow$, and two inputs I_1, I_2 such that $I_1{=_{NS}}I_2$. Non-interference of P under $\hookrightarrow$ means

that $\forall n \geq 0$, if $(P, I_1) \hookrightarrow^n (I_1', O_1)$ then $(P, I_2) \hookrightarrow^n (I_2', O_2)$, where $I_1 =_{NS} I_2$ and $O_1 =_{NS} O_2$ hold.

Note that this definition implies $len(I_1') = len(I_2')$ and $len(O_1') = len(O_2')$. This means that as long as non-sensitive entries in input ADMs do not change, the timing of syscall occurrences, which is observable by an adversary, remains the same.

Theorem 1 (**Non-interference under PaveBox**)**.** Any program P is non-interferent under $\overset{f}{\dashrightarrow}$, where f is any function that specifies the execution order.

Recall from §6.3 that the synchronized global semantics ($\rightsquigarrow$) is a specific case of $\overset{f}{\dashrightarrow}$. Thus, this theorem subsumes the non-interference under $\rightsquigarrow$ as well.

Next, I formally define the functionality preservation property of our PaveBox model. While PaveBox can restrict the behavior of a malicious ad program, it should not amend or limit the functionality of a program if it is non-inteferent under the standard execution.

Theorem 2 (**Functionality Preservation**)**.** Assume a program P that is non-interferent under $\rightarrow$. Now, $\forall n \geq 0$, if $(P, I) \rightarrow^n (I_1, O_1)$, then $(P, I) \rightsquigarrow^n (I_2, O_2)$ holds where $O_1 = O_2$.

6.5 Proof of Security Properties

6.5.1 Proof of Non-Interference

First, I introduce equivalence over the low fields (or high fields). For faceted values $v = \langle v_L, v_H \rangle$ and $v' = \langle v_L', v_H' \rangle$, I say $v =_L v'$ iff $v_L = v_L'$. Also, I say $v =_H v'$ iff $v_H = v_H'$. I extend this to faceted ADMs a and a'. I say $a =_L a'$ iff

$\forall k \in dom(a) \cup dom(a')$, $a(k) =_L a'(k)$. I also extend $=_H$ to faceted ADMs in the same way.

From the definition of $=_{NS}$, $=_L$, and $fct()$, I can show that if $a =_{NS} a'$, then $fct(a) =_L fct(a')$ holds. For $k \in K_{NS}$, $fct(a)(k) = \langle a(k), a(k) \rangle$ and $fct(a')(k) = \langle a'(k), a'(k) \rangle$. Also, I know $a(k) = a'(k)$ from $a =_{NS} a'$. Thus, $fct(a)(k) =_L fct(a')(k)$ holds for $k \in K_{NS}$. For $k \in K_S$, $fct(a)(k) = \langle v_{def}(k), a(k) \rangle$ and $fct(a')(k) = \langle v_{def}(k), a'(k) \rangle$, so $fct(a)(k) =_L fct(a')(k)$ holds. Thus, I can conclude that $fct(a) =_L fct(a')$. Similarly, I can also show that if $a =_L a'$, then $ext(a) =_{NS} ext(a')$ holds.

Now I introduce two lemmas to prove Theorem 1.

Lemma 1 (**Equivalence in Low Execution**)**.** Let us assume two transitions, $\langle \langle c, m \rangle_L, a_1, I_1, O_1 \rangle \Rightarrow \langle \langle c'_1, m'_1 \rangle_L, a'_1, I'_1, O'_1 \rangle$ and $\langle \langle c, m \rangle_L, a_2, I_2, O_2 \rangle \Rightarrow \langle \langle c'_2, m'_2 \rangle_L, a'_2, I'_2, O'_2 \rangle$, where $a_1 =_L a_2$, $I_1 =_{NS} I_2$, and $O_1 =_{NS} O_2$. Then, $c'_1 = c'_2$, $m'_1 = m'_2$, $a'_1 =_L a'_2$, $I'_1 =_{NS} I'_2$, and $O'_1 =_{NS} O'_2$ hold.

Lemma 2 (**Confinement of High Execution**)**.** Let us assume a transition $\langle \langle c, m \rangle_H, a, I, O \rangle \Rightarrow \langle \langle c', m' \rangle_H, a', I', O' \rangle$. Then, $a =_L a'$, $I = I'$, and $O = O'$ hold.

Intuitively, Lemma 1 means that if two low execution states are equivalent over the low fields and non-sensitive entries, then this equivalence is maintained after a low execution step. Lemma 2 states that a high execution step cannot break this equivalence as well.

Proof of Lemma 1. I can prove this lemma by examining the low execution's semantic rules in Figure 6.3. First, I can see the transition of a local state $\langle c, m \rangle_L$ is fully decided by this local state itself and the low fields of a_i, where i can be 1 or 2. Since $a_1 =_L a_2$, I can prove $c'_1 = c'_2$ and $m'_1 = m'_2$.

Next, I consider the semantic rules that can affect the global state a_i, I_i, and O_i. To start with, when c is '`add` x `to` k', it can update the low fields

in a_i. Still, I know that a_1 and a_2 will always be updated with the same value, $m(x)$. Therefore, $a'_1 =_L a'_2$ holds when c is an `add` command. Next, a `recv` command can also update the global state. Recall that if $a =_{NS} a'$, then $fct(a) =_L fct(a')$ holds. With this property and $I_1 =_{NS} I_2$, I can show $a'_1 =_L a'_2$ and $I'_1 =_{NS} I'_2$ hold by examining rule (L1). Similarly, I have shown that if $a =_L a'$, then $ext(a) =_{NS} ext(a')$ holds. Thus, when c is `send`, I can use this property and $a_1 =_L a_2$ to show that $O'_1 =_{NS} O'_2$ by examining rule (L3).

Proof of Lemma 2. I can also prove this lemma by examining the high execution's semantic rules in Figure 6.3. In the high execution, the semantic rules prevent any update on I and O. Also, rule (L7) only allows updates to the high fields of a. Therefore, $a =_L a'$ holds. Consequently, Lemma 2 holds.

Proof of Theorem 1. Suppose two inputs I_1 and I_2 such that $I_1 =_{NS} I_2$ holds. I will prove the following property that subsumes Theorem 1: $\forall n \geq 0$, if $\langle S_0, S_0, a_0, I_1, O_0, 0\rangle \overset{f}{\dashrightarrow}^n \langle L'_1, H'_1, a'_1, I'_1, O'_1, n\rangle$ then $\langle S_0, S_0, a_0, I_2, O_0, 0\rangle \overset{f}{\dashrightarrow}^n \langle L'_2, H'_2, a'_2, I'_2, O'_2, n\rangle$, where $L'_1 = L'_2$, $a'_1 =_L a'_2$, $I'_1 =_{NS} I'_2$, and $O'_1 =_{NS} O'_2$. Here, S_0 is the initial local state $\langle P, m_0\rangle$, where P in an input program. First, this property trivially holds when $n = 0$. Next, I prove that if this property holds for $n = k$, it also holds for $n = k + 1$. When $f(k) = \mathtt{L}$, Lemma 1 shows that $L'_1 = L'_2$, $a'_1 =_L a'_2$, $I'_1 =_{NS} I'_2$, and $O'_1 =_{NS} O'_2$ hold for $n = k+1$. Meanwhile when $f(k) = \mathtt{H}$, I can first show that $L'_1 = L'_2$ holds for $n = k + 1$ from rule (G2') in Figure 6.4. Similarly, Lemma 2 states that I/O ADM lists do not change, so $I'_1 =_{NS} I'_2$ and $O'_1 =_{NS} O'_2$ hold for $n = k + 1$. In addition, Lemma 2 shows that $=_{NS}$ holds between the two a'_1 when $n = k$ and $n = k+1$. Also, the same holds for a'_2. Thus, $a'_1 =_{NS} a'_2$ holds for $n = k + 1$, by the transitivity of $=_{NS}$. Therefore, the property is proved by induction on n.

6.5.2 Proof of Functionality Preservation

First, I define correspondence between a non-faceted ADM and a faceted ADM. Assume a raw value v and a faceted value $v' = \langle v_L, v_H \rangle$. I say $v \simeq_L v'$ iff $v = v_L$ and $v \simeq_H v'$ iff $v = v_H$. Next, I define $\simeq_L$ and $\simeq_H$ for a non-faceted ADM a and a faceted ADM a'. I say $a \simeq_L a'$ iff $\forall k \in dom(a) \cup dom(a'), a(k) \simeq_L a'(k)$. And I say $a \simeq_H a'$ iff $\forall k \in dom(a) \cup dom(a'), a(k) \simeq_H a'(k)$.

Next, I define a replacement function for sensitive entries of an ADM. I define $rep(k, v)$ as $v_{def}(k)$ if $k \in K_S$ and v if $k \in K_{NS}$. Then, I extend rep for an ADM a, by defining $rep(a)$ as $\{(k, v') | (k, v) \in a, v' = rep(k, v)\}$. I can also extend this for an ADM list.

From the definition of $\simeq_L$ and $rep()$, I can prove that $rep(a) \simeq_L fct(a)$. For $k \in K_{NS}$, $rep(a)(k) = a(k)$ and $fct(a)(k) = \langle a(k), a(k) \rangle$, so $rep(a)(k) \simeq_L fct(a)(k)$ holds. For $k \in K_S$, $rep(a)(k) = v_{def}(k)$ and $fct(a)(k) = \langle v_{def}(k), a(k) \rangle$, so $rep(a)(k) \simeq_L fct(a)(k)$ holds. Similarly, I can also prove $a \simeq_H fct(a)$.

In addition, I can show that if $a \simeq_L a'$ then $a =_{NS} ext(a')$. From $a \simeq_L a'$, I know that the low field of $a'(k)$ is equal to $a(k)$. Since $ext()$ chooses these low fields for $k \in K_{NS}$, $a =_{NS} ext(a')$ holds. In a similar way, I can show that if $a \simeq_H a'$ then $a =_S ext(a')$.

Lastly, I define $=_{sys}$ on commands c_1 and c_2. First, I say $c_1 =_{recv} c_2$ iff (i) $c_1 \neq$ `recv` and $c_2 \neq$ `recv` or (ii) $c_1 = c_2 =$ `recv`. Also, I say $c_1 =_{send} c_2$ iff (i) $c_1 \neq$ `send` and $c_2 \neq$ `send` or (ii) $c_1 = c_2 =$ `send`. Finally, I say $c_1 =_{sys} c_2$ iff $c_1 =_{recv} c_2$ and $c_1 =_{send} c_2$.

Now I introduce two lemmas to prove Theorem 2.

Lemma 3 (**Correspondence to Low Execution**)**.** Let us assume a transition step with standard semantics $\langle c, m, a_1, I_1, O_1 \rangle \rightarrow \langle c'_1, m'_1, a'_1, I'_1, O'_1 \rangle$ and a global transition step with the PAVEBOX semantics $\langle \langle c, m \rangle, \langle c_H, m_H \rangle, a_2, I_2, O_2 \rangle \rightsquigarrow \langle \langle c'_2, m'_2 \rangle, H', a'_2, I'_2, O'_2 \rangle$. Now, if $a_1 \simeq_L a_2$, $I_1 = rep(I_2)$ and $O_1 =_{NS} O_2$, then

$c'_1 = c'_2$, $m'_1 = m'_2$, $a'_1 \simeq_L a'_2$, $I'_1 = rep(I'_2)$, and $O'_1 =_{NS} O'_2$.

Lemma 4 (**Correspondence to High Execution**). Let us assume a transition step with standard semantics $\langle c, m, a_1, I_1, O_1\rangle \rightarrow \langle c'_1, m'_1, a'_1, I'_1, O'_1\rangle$ and a global transition step with the PAVEBOX semantics $\langle\langle c_L, m_L\rangle, \langle c, m\rangle, a_2, I_2, O_2\rangle \rightsquigarrow \langle L', \langle c'_2, m'_2\rangle, a'_2, I'_2, O'_2\rangle$. Now, if $a_1 \simeq_H a_2$, $I_1 =_S I_2$, $O_1 =_S O_2$, and $c =_{sys} c_L$, then $c'_1 = c'_2$, $m'_1 = m'_2$, $a'_1 \simeq_H a'_2$, $I'_1 =_S I'_2$, and $O'_1 =_S O'_2$.

Intuitively, Lemma 3 means that if there is a correspondence between a standard execution and a low execution, it is maintained after a single step. Lemma 4 describes the preservation of the correspondence between a standard execution and a high execution.

Proof of Lemma 3. With rule (G1) of Figure 6.3 and previously proved Lemma 2, I can decompose the global transition step in Lemma 3 into the following two local transition steps: (i) $\langle\langle c, m\rangle_L, a_2, I_2, O_2\rangle \Rightarrow \langle\langle c'_2, m'_2\rangle_L, \tilde{a}_2, I'_2, O'_2\rangle$ and (ii) $\langle\langle c_H, m_H\rangle_H, \tilde{a}_2, I'_2, O'_2\rangle \Rightarrow \langle\langle c'_H, m'_H\rangle_H, a'_2, I'_2, O'_2\rangle$.

First, I consider the low execution step (i). I know that the transition of c and m in the standard execution is decided by c, m, and a_1. Meanwhile, in the low execution, the transition of $\langle c, m\rangle$ is decided by c, m, and the low fields of a_2. From $a_1 \simeq_L a_2$, I can see that c, m in the standard execution and c, m in the low execution always change in the same way. Thus, $c'_1 = c'_2$ and $m'_1 = m'_2$ hold.

Next, I will prove that $a'_1 \simeq_L \tilde{a}_2$, $I'_1 = rep(I'_2)$, and $O'_1 =_{NS} O'_2$. Note that if c does not modify ADM components (i.e., $a_1, a_2, I_1, I_2, O_1, O_2$), then these properties are directly satisfied. Thus, I will consider commands that can modify ADM components. First, when c is '`add` x `to` k', the standard execution updates a_1 with $m(x)$ while the low execution updates a low field of a_2 with $m(x)$. Therefore, $a'_1 \simeq_L a'_2$ holds. Next, let us assume c is `recv`. Recall that I have shown $rep(a) \simeq_L fct(a)$. With this property and $I_1 = rep(I_2)$, I can prove

that $a'_1 \simeq_L \tilde{a}_2$ from rule (1) and (L1). Also, these rules remove the first element in I_1 and I_2 respectively, so $I'_1 = rep(I'_2)$ holds, too. Lastly, when c is `send`, I will use the fact that if $a \simeq_L a'$ then $a =_{NS} ext(a')$ holds. With this property and $a_1 \simeq_L a_2$, I can show that $O'_1 =_{NS} O'_2$ holds from rule (2) and (L3).

Now, I move on to the high execution step (ii). From Lemma 2, $\tilde{a}_2 =_L a'_2$ holds. Since I have proven $a'_1 \simeq_L \tilde{a}_2 =_L a'_2$, I can see that $a'_1 \simeq_L a'_2$.

Proof of Lemma 4. As I did in the proof of Lemma 3, I split the transition with global semantics into two local transitions: (i) $\langle\langle c_L, m_L\rangle_L, a_2, I_2, O_2\rangle \Rightarrow \langle\langle c'_L, m'_L\rangle_L, \tilde{a}_2, I'_2, O'_2\rangle$ and (ii) $\langle\langle c, m\rangle_H, \tilde{a}_2, I'_2, O'_2\rangle \Rightarrow \langle\langle c'_2, m'_2\rangle_H, a'_2, I'_2, O'_2\rangle$.

I start by proving $I'_1 =_S I'_2$ and $O_1 =_S O'_2$. First, I will prove $I'_1 =_S I'_2$. When c is not `recv`, neither is c_L, from $c =_{sys} c_L$. Then, $I_1 = I'_1$ and $I_2 = I'_2$, so $I'_1 =_S I'_2$. When c is `recv`, so is c_L, and the first element is removed respectively from I_1 and I_2. Since $I_1 =_S I_2$, I can see $I'_1 =_S I'_2$. Next, I will prove $O_1 =_S O'_2$. When c is not `send`, neither is c_L, from $c =_{sys} c_L$. Then, $O_1 = O'_1$ and $O_2 = O'_2$, so $O'_1 =_S O'_2$. When c is `send`, so is c_L, and I will use the fact that if $a \simeq_H a'$ then $a =_S ext(a')$. With this property and $a_1 \simeq_H a_2$, I can see $O'_1 =_S O'_2$.

Now, I will prove $c'_1 = c'_2$, $m'_1 = m'_2$ and $a'_1 \simeq_H a'_2$. For this, I should consider $c =$ `recv` case separately. When $c =$ `recv`, so is c_L, from $c =_{sys} c_L$. Recall that I have shown $a \simeq_H fct(a)$. Using this property, I can examine rule (1) and (L1) to conclude that $a'_1 \simeq_H \tilde{a}_2$ holds. In addition, from rule (L2), $\tilde{a}_2 = a'_2$. From $a'_1 \simeq_H \tilde{a}_2 = a'_2$, I can see $a'_1 \simeq_H a'_2$. Also, $c'_1 = c'_2$ and $m'_1 = m'_2$ trivially hold from rule (1) and (L2).

Next, I consider the case where c is not `recv`. Then, c_L is not `recv` as well, so the low execution can only update the low fields of a_2, and $a_2 =_H \tilde{a}_2$ holds. From $a_1 \simeq_H a_2 =_H \tilde{a}_2$, I can see $a_1 \simeq_H \tilde{a}_2$. Now I can prove $c'_1 = c'_2$ and $m'_1 = m'_2$. This time, the transition of $\langle c, m\rangle$ is decided by c, m, and the high fields of $\tilde{a}_2$. From $a_1 \simeq_H \tilde{a}_2$, I can see that c, m in the standard execution and c, m in the high execution always change in the same way. Therefore, $c'_1 = c'_2$ and

$m'_1 = m'_2$ hold. Lastly, I prove $a'_1 \simeq_H a'_2$. When c is '`add` x `to` k', the standard execution updates a_1 with $m(x)$ while the low execution updates a high field of $\tilde{a}_2$ with m(x). Thus, $a'_1 \simeq_H a'_2$ holds. If c is not `add`, $a_1 = a'_1$ and $\tilde{a}_2 = a'_2$, so $a'_1 \simeq_H a'_2$ directly holds from $a_1 \simeq_H \tilde{a}_2$.

Proof of Theorem 2. Assume input I, non-interferent program P, and the following three executions. First, I assume a standard execution with I: $\langle P, m_0, a_0, I, O_0\rangle \rightarrow^n \langle c_1, m_1, a_1, I_1, O_1\rangle$. Second, I assume another standard execution with $rep(I)$: $\langle P, m_0, a_0, rep(I), O_0\rangle \rightarrow^n \langle c_2, m_2, a_2, I_2, O_2\rangle$. Third, I assume a PAVEBOX execution with I: $\langle S_0, S_0, a_0, I, O_0\rangle \rightsquigarrow^n \langle\langle c_L, m_L\rangle, \langle c_H, m_H\rangle , a_3, I_3, O_3\rangle$.

To prove Theorem 2, I should prove that $\forall n \geq 0$, $O_1 = O_3$. I can prove this by showing that (i) $\forall n \geq 0$, $O_1 =_{NS} O_3$, and (ii) $\forall n \geq 0$, $O_1 =_S O_3$.

First, I prove (i) $\forall n \geq 0$, $O_1 =_{NS} O_3$. I first use the non-interference of P. Since P is non-interferent and $rep(I) =_{NS} I$, I know that $\forall n \geq 0$, $O_1 =_{NS} O_2$ holds from Definition 1. Next, I will prove that $O_2 =_{NS} O_3$. For this, I prove that $\forall n \geq 0$, $c_2 = c_L$, $m_2 = m_L$, $a_2 \simeq_L a_3$, $I_2 = rep(I_3)$, and $O_2 =_{NS} O_3$. When $n = 0$, this property trivially holds. Also, if the property holds for $n = k$, I can prove that it also holds for $n = k+1$, using Lemma 3. Therefore, the property holds $\forall n \geq 0$, by induction on n. At this point, I have shown $O_1 =_{NS} O_2 =_{NS} O_3$.

Next, I prove (ii) $\forall n \geq 0$, $O_1 =_S O_3$. For this, I prove that $\forall n \geq 0$, $c_1 = c_H$, $m_1 = m_H$, $a_1 \simeq_H a_3$, $I_1 =_S I_3$, and $O_1 =_S O_3$. When $n = 0$, this property trivially holds. Next, I will use Lemma 4 to show that if the property holds for $n = k$, then it also holds for $n = k + 1$. However, I must first show that $\forall n \geq 0$, $c_1 =_{sys} c_L$ holds. From the non-interference of P, I know that $\forall n \geq 0$, $c_1 =_{sys} c_2$, as the length of I_1, I_2, O_1 and O_2 can change only by `recv` and `send`. Also, during the proof of $O_2 =_{NS} O_3$, I have proved $\forall n \geq 0$, $c_2 = c_L$. I now know that $\forall n \geq 0$, $c_1 =_{sys} c_L$ holds, so I can use Lemma 4 and prove the property with induction on n.

$$(L1)\quad \frac{I = [a_1, a_2, ..., a_n] \quad I' = [a_2, ..., a_n]}{\langle\langle \texttt{recv}, m\rangle_L, a, I, O\rangle \Rightarrow \langle\langle \texttt{skip}, m\rangle_L, fct(a_1), I', O\rangle}$$

$$(L2)\quad \frac{}{\langle\langle \texttt{recv}, m\rangle_H, a, I, O\rangle \Rightarrow \langle\langle \texttt{skip}, m\rangle_H, a, I, O\rangle}$$

$$(L3)\quad \frac{O = [a_1, ..., a_n] \quad O' = [a_1, ..., a_n, ext(a)]}{\langle\langle \texttt{send}, m\rangle_L, a, I, O\rangle \Rightarrow \langle\langle \texttt{skip}, m\rangle_L, a, I, O'\rangle}$$

$$(L4)\quad \frac{}{\langle\langle \texttt{send}, m\rangle_H, a, I, O\rangle \Rightarrow \langle\langle \texttt{skip}, m\rangle_H, a, I, O\rangle}$$

$$(L5)\quad \frac{a(k) = \langle v_L, v_H\rangle \quad m' = m[x \mapsto v_l]}{\langle\langle \texttt{get } k \texttt{ to } x, m\rangle_l, a, I, O\rangle \Rightarrow \langle\langle \texttt{skip}, m'\rangle_l, a, I, O\rangle}$$

$$(L6)\quad \frac{m(x) = v \quad a(k) = \langle v_L, v_H\rangle \quad a' = a[k \mapsto \langle v, v_H\rangle]}{\langle\langle \texttt{add } x \texttt{ to } k, m\rangle_L, a, I, O\rangle \Rightarrow \langle\langle \texttt{skip}, m\rangle_L, a', I, O\rangle}$$

$$(L7)\quad \frac{m(x) = v \quad a(k) = \langle v_L, v_H\rangle \quad a' = a[k \mapsto \langle v_L, v\rangle]}{\langle\langle \texttt{add } x \texttt{ to } k, m\rangle_H, a, I, O\rangle \Rightarrow \langle\langle \texttt{skip}, m\rangle_H, a', I, O\rangle}$$

$$(L8)\quad \frac{eval(e, m) = v \quad m' = m[x \mapsto v]}{\langle\langle x := e, m\rangle_l, a, I, O\rangle \Rightarrow \langle\langle \texttt{skip}, m'\rangle_l, a, I, O\rangle}$$

$$(L9)\quad \frac{}{\langle\langle \texttt{skip}; c_2, m\rangle_l, a, I, O\rangle \Rightarrow \langle\langle c_2, m\rangle_l, a, I, O\rangle}$$

$$(L10)\quad \frac{c_1 \neq \texttt{skip} \quad \langle\langle c_1, m\rangle_l, a, I, O\rangle \Rightarrow \langle\langle c_1', m'\rangle_l, a', I', O'\rangle}{\langle\langle c_1; c_2, m\rangle_l, a, I, O\rangle \Rightarrow \langle\langle c_1'; c_2, m'\rangle_l, a', I', O'\rangle}$$

$$(L11)\quad \frac{}{\langle\langle \texttt{skip}, m\rangle_H, a, I, O\rangle \Rightarrow \langle\langle \texttt{skip}, m\rangle_H, a, I, O\rangle}$$

$$(G1)\quad \frac{\begin{array}{c}\langle\langle c_1, m_1\rangle_L, a, I, O\rangle \Rightarrow \langle\langle c_1', m_1'\rangle_L, a', I', O'\rangle \\ \langle\langle c_2, m_2\rangle_H, a', I', O'\rangle \Rightarrow \langle\langle c_2', m_2'\rangle_H, a'', I'', O''\rangle\end{array}}{\begin{array}{c}\langle\langle c_1, m_1\rangle, \langle c_2, m_2\rangle, a, I, O\rangle \rightsquigarrow \\ \langle\langle c_1', m_1'\rangle, \langle c_2', m_2'\rangle, a'', I'', O''\rangle\end{array}}$$

Figure 6.3: Local and global semantics under PaveBox are defined.

$$(G1') \quad \frac{f(n) = L \;\; \langle\langle c_1, m_1\rangle_L, a, I, O\rangle \Rightarrow \langle\langle c_1', m_1'\rangle_L, a', I', O'\rangle}{\langle\langle c_1, m_1\rangle, \langle c_2, m_2\rangle, a, I, O, n\rangle \xrightarrow{f} \langle\langle c_1', m_1'\rangle, \langle c_2, m_2\rangle, a', I', O', n+1\rangle}$$

$$(G2') \quad \frac{f(n) = H \;\; \langle\langle c_2, m_2\rangle_H, a, I, O\rangle \Rightarrow \langle\langle c_2', m_2'\rangle_H, a', I', O'\rangle}{\langle\langle c_1, m_1\rangle, \langle c_2, m_2\rangle, a, I, O, n\rangle \xrightarrow{f} \langle\langle c_1, m_1\rangle, \langle c_2', m_2'\rangle, a', I', O', n+1\rangle}$$

Figure 6.4: Global semantics without the synchronization is defined.

Chapter 7

Discussion

7.1 Side-channel Attacks

Preventing side-channel attacks is important; some vulnerabilities on Intel SGX are discovered. While such attacks are out of the scope in PAVE, I discuss how to mitigate the side-channel attacks.

To launch attacks through side-channels, an attacker usually runs an adversarial process either on the same core or on another core. The same-core adversary who shares the same physical core with a victim process can exploit the shared resources such as Branch Prediction Units [33, 49], L1/L2 cache [54], and Translation Lookaside Buffer (TLB) [39]. On the contrary, in the cross-core side-channel attack, an adversary does not share the same physical core with the victim process. Thus, the adversary exploits off-core resources such as last-level cache (LLC) [67] and DRAM row buffer [61]. Moreover, an untrusted OS can exploit page table entries to launch controlled-channel attacks [84, 55]. The adversary can manipulate the page table to trace accessed pages, which can be used to analyze the executions of the victim program.

Lastly, microarchitectural vulnerabilities should be addressed such as speculative and transient executions [81, 20, 47, 79, 80].

Even if there is no silver-bullet solution, multiple mitigations can be used [21, 59, 70, 41, 3, 73, 22, 71, 60]. First of all, the same-core side-channel attacks [33, 49, 54, 39, 39] can be mitigated by disabling the adversarial process residing in the same core. Hyperrace [21], Varys [59] and Déjá Vu [22] make two honest threads run in the same core. Or, it can be useful to disable hyperthreading; remote attestation can detect whether hyperthreading is disabled. I applied Hyperrace to PAVE by instrumenting its co-location test for every basic block and it showed that the overhead was modest (under 5%). Also, many vulnerabilities [81, 20, 47, 79] can be mitigated by microcode patch [58], which is necessary to keep its version up-to-date. The patch of hardware can be attested by the user agent through the remote attestation. Lastly, the PAVE toolchain can provide compiler-based mitigations [73, 70, 22, 3] or application-based approaches [41, 71, 60].

Furthermore, in advertising, the benefit of side-channel attacks is marginal. Firstly, it is difficult to target a particular user. An ad request is triggered by a user and the ad servers usually cannot predict which user request will be processed. It means that targeting a particular user would be impractical. Also, a single enclave thread in PAVE processes only one user profile at a time, which decreases the utility of the information to be leaked. Secondly, the performance of advertising will be reduced for the attackers (i.e., ad servers) if they launch the side-channel attacks. Sometimes, the attackers isolate cores or caches to reduce the noise from the attacks, which degrades the performance. Thus, the attackers may not be motivated to collect the profiles of random users. Further, many advertising services rely on a cloud [38]. If PAVE is serviced as a server-less solution in a third-party cloud, the side-channel attacks would be infeasible.

7.2 After AD Selections

PAVE enables the user agent to fetch a tailored ad URL during the ad selection. Note that the ad delivery of retrieving the raw ad material using the ad URL can invade the user privacy. The history of ads served to a specific IP address can be used to infer the user interests. For this, prior studies [64, 42, 8] use a proxy [64, 42] or a privacy-preserving retrieval technique [8] to anonymize the user agent. Also, it is possible to use a third-party CDN for serving ads [76].

For billing and measuring the effectiveness of the ads that are delivered, there already exists an API to aggregate the ad-related information in a privacy-preserving fashion [23]. Also, some proposals rely on a third-party service such as IAB (Interactive Advertising Bureau) [8, 42, 76]. PAVE assumes such techniques for anonymized ad retrievals and privacy-preserving billing/measurements.

Chapter 8

Evaluation

8.1 Experiment Setup

Prototype. I implement a PaveBox prototype with the toolchain by extending NaCl and using LLVM. Currently, our toolchain supports C/C++ language with additional libraries for libc and the FS interfaces. It also instruments the code sequence for synchronization, which is inserted at the beginning of each basic block of the ad program. To narrow the software interface, the PaveBox manages virtual memory lists for memory mapping (i.e., mmap, brk, and munmap), process information (i.e., getpid), and random numbers (i.e., getrandom) inside an enclave without host syscalls. Furthermore, to reduce the overhead of (re-)creating a new execution, the PaveBox provides a snapshot that stores its state after initializing an ad program and restores the snapshot right after finishing an ad request. Additionally, to thwart Iago attacks [19], the PaveBox validates inputs from the untrusted OS.

I implement three different versions of the PaveBox; Pave Single, Pave NoSync, and Pave Sync. Pave Single does not execute the high execution,

which means that there is only a low execution. In PAVE NoSync, while shadow execution is implemented, the synchronization is omitted. Note that it still preserves non-interference. PAVE Sync fully implements the PAVEBOX and synchronizes the high and low executions.

Benchmark. I implemented ad programs for three ad selection scenarios: decision tree (DT), content-based filtering (CF), and real-time bidding (RTB). The first and second scenarios are for a single ad network. The decision tree represents ad campaigns that advertisers register. Given user interests, the ad network deterministically responds an ad. For the second scenario, I use a content-based filtering recommendation scheme with TF-IDF (Term Frequency-Inverse Document Frequency). It is one of the widely used algorithms in advertising [37]. The third scenario is RTB, which consists of one ADX with an SSP and two DSPs. Here, the SSP and the ADX are co-located to simplify the experiment since processing the user profile is mainly done by the DSPs. The two DSPs each choose a bid by the two different bidding algorithms in [87]. The user agent sends her profile based on the IPinYou dataset [51]. Note that the tasks for the ad program are matching or arithmetic operations of the above ad models. That is, the models are trained beforehand, and their computations are done by general Linux programs (without SGX).

Environments. I measure (i) the computation overhead to initialize the PAVEBOX, (ii) the user latency to receive an ad URL, and (iii) the execution time of an ad program. For comparison purposes, baseline programs are general Linux programs without the enclave or the PAVEBOX. Every plot is averaged over ten runs. I conduct experiments on a Linux desktop computer equipped with an Intel Core i7-8700 CPU with 6 cores and 12 hyper-threads. As all the ad programs and the user agent run on the same machine, the network propagation delay is not reflected. I measured the four different ADM sizes (i.e., 2,048, 4,096, 8,192, and 16,384 bytes), but there were no notable

differences. Thus, I plot the results when the ADM is 8,192 bytes.

8.2 PaveBox Initialization Overhead

To evaluate the computation overhead to initialize and load the PaveBox, I measure the time to run a simple C test program '`int main{return 0;}`.' Specifically, I measure (i) the time to create and load an enclave memory, (ii) the time to initialize the PaveBox after entering the enclave, and (iii) the time to run the test program and terminate the enclave. It takes 1,515 ms to create and load an enclave memory. Initializing the PaveBox takes 1,552 ms since it includes the communication with Intel Attestation Service for remote attestation (to be explained in §5.4). The time for running the test program is 2 ms. The total time from creating the enclave to exiting from the enclave is 3,069 ms, which is the minimum time to run a simple program in the PaveBox. Note that, by using the snapshot technique, the initialization overhead ((i) and (ii)) incurs only once when the ad server starts the ad program. In our evaluation, the snapshot overhead is 3.7 ms, which is much more efficient than the initialization.

8.3 User Latency

The user latency means the time between the moment of sending an ad request and that of receiving an ad URL, as shown in Figure 8.1. While the ad network model has only one ad server, an RTB is performed by three ad servers. The delays of the ad network and RTB models are approximately proportional to the complexity of the interactions between the ad servers. The user latency increases from the baseline to Pave Sync. Compared to the baseline, the user latency for Pave Single is increased by 21.8% on average. In addition, shadow execution (i.e., in Pave NoSync) and the synchronization (i.e., in Pave Sync)

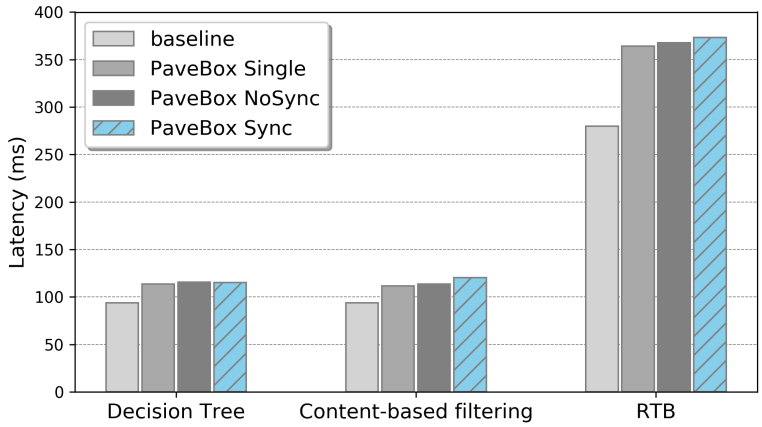
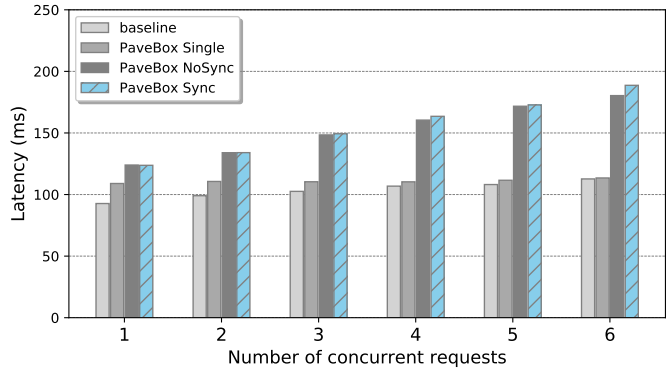


Figure 8.1: The user latency to receive an ad URL is plotted.

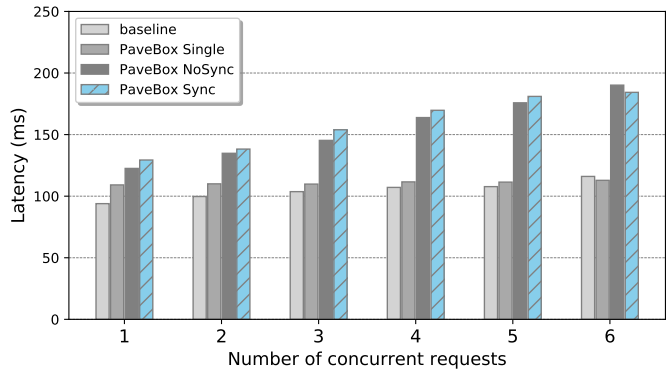
incur 3.0% and 2.4% overheads, respectively.

Also, to measure the impact on the user latency when the PaveBox handles multiple ad requests at the same time, I increase the number of execution pairs from 1 to 6. Note that since SGX1 limits the number of threads to the number of CPU cores, I evaluate the DT and CF scenarios relying on one single server. Figure 8.2 shows the user latency as the number of execution pairs varies. As the number of execution pairs is incremented by one, the user latency of Pave increases more than that of baseline. I believe it is due to the limited size of EPC. Also, as the number of execution pairs is incremented by one, the number of SGX threads of Pave NoSync and Pave Sync is incremented by two. On the other hand, that of Pave Single is incremented by one. Therefore, the user latencies of Pave NoSync and Pave Sync increases more than that of Pave Single.

Although Pave incurs some overhead, it may not harm the user experiences notably. Current ad companies provide asynchronous ads (e.g., Amazon Associates [6]), which loads/renders non-ad content before ads are delivered. That is, users tend to focus on the non-ad content of webpages, and hence



(a) Decision Tree.

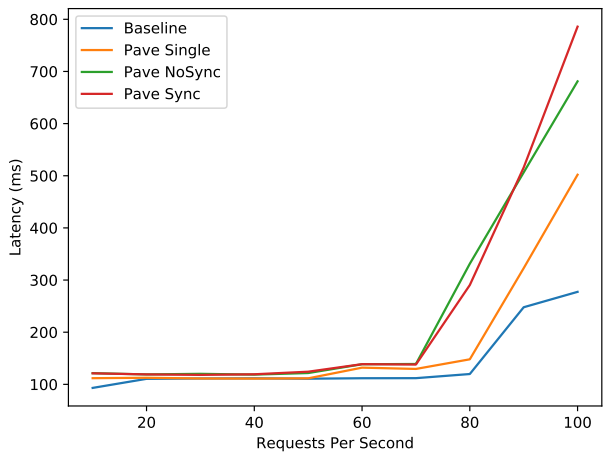


(b) Content-based Filtering.

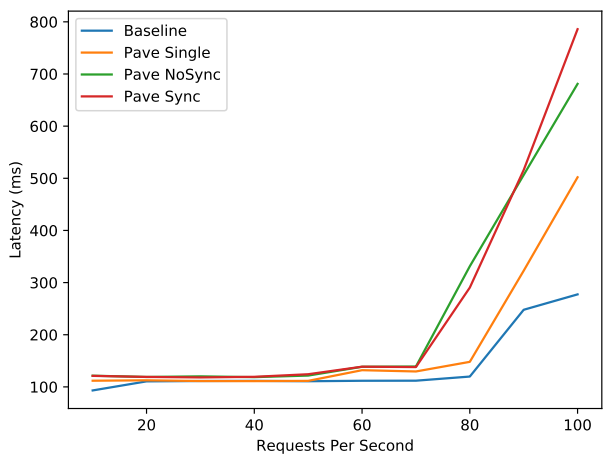
Figure 8.2: The user latency is measured when the number of concurrent ad requests varies.

they may not notice whether rendering ads is slightly more delayed.

Further, I measure the user latency when request rate increases to represent the impact on the server throughput when Pave is deployed. As illustrated in Figure 8.3, regardless of the application scenarios, the user latencies soar after request rate becomes 70 requests per second in Pave NoSync and Pave Sync and 80 requests per second in Pave Single. On the other hand, the user latency of the baseline is stable compared to Pave. It means that Pave can



(a) Decision Tree.



(b) Content-based Filtering.

Figure 8.3: The user latency is measured when the number of ad requests per second varies.

handle less requests than baseline. It is mainly due to the limited number of threads of SGX.

Table 8.1: The execution time and the number of page faults of each ad program are plotted. PP refers the pre-processing delay and AP refers the ad-processing delay. PF refers the total number of page faults. The unit for PP and AP is ms.

			Baseline	PAVE Single	PAVE NoSync	PAVE Sync
DT		PP	7.8	16.9	17.6	21.6
		AP	93.7	110.3	115.5	115.1
		PF	1,265	3,023	3,838	3,872
CF		PF	7.9	17.4	18.9	26.4
		AP	93.4	109.8	113.4	120.3
		FF	1,262	3,017	3,831	3,837
RTB	**DSP1**	PF	12.5	29.7	31.3	47.2
		AP	92.0	113.5	117.4	125.9
		FF	348	1,539	1,703	1,740
	DSP2	PF	12.0	27.7	27.4	42.9
		AP	92.9	129.0	132.9	132.4
		FF	450	1,535	1,699	1,738
	SSP&ADX	PF	0.1	0.1	0.1	0.1
		AP	279.8	363.8	367.5	373.1
		FF	268	1,230	1,279	1,290

8.4 Execution Time

Table 8.1 shows the execution time to run an ad program, which consists of (i) the pre-processing delay to initialize data until it is ready for an incoming connection, and (ii) the ad processing delay from the arrival of an ad request to the termination. Also, the total number of page faults is measured to see the paging overhead from the limited EPC size of 128 MB.

For the pre-processing delay, shadow execution (i.e., PAVE NoSync) and synchronization (i.e., PAVE Sync) incur 3.4% and 34.0% overheads compared to

PAVE Single and PAVE NoSync, respectively. On the other hand, PAVE Single incurs 101.0% overhead compared to the baseline. It is mainly due to the SGX overhead to access the enclave memory and the code overhead of the SFI-compliant binary. Especially, the pre-processing requires the repetitive steps to read the trained model from the file system and to initialize related data, which incurs heavy overhead. Nevertheless, the pre-processing is performed only once in advance, and only the ad processing part will be repeated for each incoming ad request. Note that the ad processing time is not substantial in all the scenarios.

Chapter 9

Related Work

9.1 Privacy-preserving Ad Systems

Several studies have focused on the targeted advertising systems while preserving user privacy. In Privad [42] and Adnostic [76], a user agent builds a user profile. The user agent prefetches k ads from an ad network based on a coarse-grained interest category [42] or the context of the publisher website [76]. Then, the agent selects one of them based on the user profile. AdScale [40] improves the scalability of Adnostic by using cryptographic voting schemes. Hardt and Nath [43] formalize a model that balances user privacy, the computation overheads, and the ad utility. However, the ad utility is decreased since only the coarse-grained interest category or the context of websites is used to fetch ads. Also, the above approaches constrain current ad selection mechanisms.

In Obliviad [8], the user agent makes a user profile, which is sent to an ad network equipped with trusted hardware. The ad server selects the best keyword that matches the user profile. Using the oblivious RAM (ORAM)

technique, the ad server "obliviously" retrieves an ad that matches the keyword from its file system. Obliviad focuses on privately retrieving the raw ad material for the given keyword.

Reznichenko et al. [64] suggest a privacy-preserving ad ranking system. A user agent computes a score for each candidate ad using the information about the ad and the user interests profiled locally. Likewise, the ad network computes a score based on the metadata (of an ad) such as its bid without the user information. Two scores are aggregated by any of three entities (the user agent, the ad network, or a third-party) to serve the ad of the highest final score. It is limited to a static ranking system rather than RTB.

Pri-RTB [28] leverages homomorphic encryption to compute a bid price on encrypted user profiles. It supports only additions, which limits the ad selection mechanisms. A secure multi-party computation based approach [78] has the same issue.

Google Chrome suggests Topics [36] and FLEDGE [35]. In Topics, a browser classifies a URL to topics from 350 predetermined categories. Then, ad companies can retrieves the topics and selects an appropriate ad based on it. FLEDGE [35] enables an on-browser auction. Whenever a browser visits an advertiser's site, it fetches a code for an auction logic. Then, when the browser makes an ad request at the publisher's site, it selects an ad based on the auction logic codes. Specifically, Topics is for interest-based advertising and FLEDGE is for retargeted advertising.

Brave [14] runs its own ad platform. Brave browser applies machine learning models based on user behaviors on a local browser. The models and a list of ads are periodically downloaded from a Brave ad server.

Additionally, AdAttester [50] uses trusted hardware to prevent click fraud attacks in advertising.

9.2 Information Flow Control

As IFC has been studied comprehensively, I focus on dynamic IFC systems. I first introduce IFC studies that are not based on SME. In decentralized IFC approaches using a process-level IFC [48, 86, 85], files and processes are assigned labels to specify whether a process has a privilege to access a file with a particular tag. The label and the tag are tracked by a kernel module.

Thoth [30], Riverbed [83], and Mitigator [52] control user data under a user-defined policy at a server. They aim to thwart poorly-designed or buggy programs that inadvertently leak the user data. In Thoth, a kernel-level monitor that tracks data flows through I/O interceptions enforces the policy. In Riverbed, a taint-tracking instrument similar to TaintDroid [31] checks the policy for a Python source code. In Mitigator, a verifier enclave checks the compliance of the source code with a privacy policy using static analysis and produces a signature. Riverbed and Mitigator leverage the trusted hardware for the remote attestation.

The above schemes target an adversarial program running on a trusted server. In an untrusted server, Ryoan [44] builds a trusted sandbox by adapting the NaCl to the Intel SGX settings. To block any implicit leakage through the syscall interface, the trusted sandbox disallows every syscall except one output of messages. (Still, their destinations are determined in advance.) Thus, it severely constrains the functionality of an untrusted program. Moreover, timing-channel leakages are not considered. The limitations preclude Ryoan from being adopted in advertising, which motivates us to propose Pave that provides more flexibility to untrusted programs.

After SME is introduced [16, 29], Kashyap et al. [45] generalize it as a scheduling approach and analyze its security from the perspective of timing- and termination-(in)sensitive non-interference. Also, declassification-aware SME

schemes are proposed [63, 12]. Rafnsson and Sabelfeld [63] use the concept of an observable channel to declassification.

Since SME requires a program of interest to be executed multiple times, some techniques try to simulate the result of the SME without multiple executions [9, 7, 66, 56, 5]. Static transformation [9] is proposed to dynamically transform a code at runtime. Similarly, Multiple Facets [7] defines evaluation rules to achieve timing-insensitive non-interference, which is further developed to Faceted SME to achieve the strong security [66, 56, 5].

While the above studies generalize and formalize SME on a well-defined model, its application and implementation pose another challenge in practice. FlowFox [25, 26] proposes the security policies for Web APIs and in-browser monitors. Pfeffer et al. [62] aim to improve the SME performance when the program is running on Unix-like systems. Also, SME is adapted for Android applications by Ariel [17].

Chapter 10

Conculsion

I propose PAVE that allows online ad companies to select an ad that matches a given user profile while preserving its privacy. In PAVE, to receive an ad impression, a user agent propagates the user profile along with a context profile (of a visiting website) to distributed ad servers. I propose shadow execution and faceted session to block any explicit or implicit leakage of the user profile while allowing ad servers to run their arbitrary ad programs. Given an ad program, a PAVE runtime monitor, dubbed PAVEBOX, monitors and controls its high and low executions, which handle the real and dummy user profiles, respectively. The PAVEBOX blocks an explicit or implicit information leakage from the high execution by comparing its behaviors to those of the low execution. Our prototype-based experiments show that the overhead of PAVE is not substantial, which demonstrates that it can be deployed in commercial services.

Bibliography

[1] G. Acar, C. Eubank, S. Englehardt, M. Juarez, A. Narayanan, and C. Diaz. The web never forgets: Persistent tracking mechanisms in the wild. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 674–689. ACM, 2014.

[2] F. AdTech. Ad tech GDPR complaint is extended to four more european regulators, May 2019. [Online]. Available: `https://fixad.tech/ad-tech-gdpr-complaint-is-extended-to-five-more-european-regulators/` (Retrieved, July 25, 2022).

[3] A. Ahmad, B. Joe, Y. Xiao, Y. Zhang, I. Shin, and B. Lee. Obfuscuro: A commodity obfuscation engine on intel sgx. In *Network and Distributed System Security Symposium*, 2019.

[4] A. Ahmad, J. Kim, J. Seo, I. Shin, P. Fonseca, and B. Lee. Chancel: efficient multi-client isolation under adversarial programs. In *Annual Network and Distributed System Security Symposium (NDSS)*, 2021.

[5] M. Algehed, A. Russo, and C. Flanagan. Optimising faceted secure multi-execution. In *2019 IEEE 32nd Computer Security Foundations Symposium (CSF)*, pages 1–115. IEEE, 2019.

[6] Amazon. Asynchronous ads, n.d. [ONLINE]. Available: `https:`

//affiliate-program.amazon.com/help/topic/t423?ac-ms-src=recofaqasyncguide (Retrieved, July 25, 2022).

[7] T. H. Austin and C. Flanagan. Multiple facets for dynamic information flow. In *Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 165–178, 2012.

[8] M. Backes, A. Kate, M. Maffei, and K. Pecina. Obliviad: Provably secure and practical online behavioral advertising. In *2012 IEEE Symposium on Security and Privacy*, pages 257–271. IEEE, 2012.

[9] G. Barthe, J. M. Crespo, D. Devriese, F. Piessens, and E. Rivas. Secure multi-execution through static program transformation. In *Formal Techniques for Distributed Systems*, pages 186–202. Springer, 2012.

[10] M. A. Bashir, S. Arshad, W. Robertson, and C. Wilson. Tracing information flows between ad exchanges using retargeted ads. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 481–496, 2016.

[11] D. Bohn. Google delays blocking third-party cookies in chrome until 2023, June 2021. [Online]. Available: https://www.theverge.com/2021/6/24/22547339/google-chrome-cookiepocalypse-delayed-2023 (Retrieved, July 25, 2022).

[12] I. Boloşteanu and D. Garg. Asymmetric secure multi-execution with declassification. In *International Conference on Principles of Security and Trust*, pages 24–45. Springer, 2016.

[13] N. Boucher. Multi-domain sfi, May 2019. [ONLINE]. Available: https://github.com/nickboucher/Multi-Domain-SFI/blob/master/paper/Multi%20Domain%20SFI.pdf (Retrieved, July 25, 2022).

[14] Brave. An introduction to brave's in-browser ads, September 2020. [ONLINE]. Available: `https://brave.com/intro-to-brave-ads/` (Retrieved, July 25, 2022).

[15] A. Cahn, S. Alfeld, P. Barford, and S. Muthukrishnan. An empirical study of web cookies. In *Proceedings of the 25th International Conference on World Wide Web*, pages 891–901. International World Wide Web Conferences Steering Committee, 2016.

[16] R. Capizzi, A. Longo, V. Venkatakrishnan, and A. P. Sistla. Preventing information leaks through shadow executions. In *2008 Annual Computer Security Applications Conference (ACSAC)*, pages 322–331. IEEE, 2008.

[17] D. Chakraborty, C. Hammer, and S. Bugiel. Secure multi-execution in android. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pages 1934–1943, 2019.

[18] F. Chanchary and S. Chiasson. User perceptions of sharing, advertising, and tracking. In *Eleventh Symposium On Usable Privacy and Security ({SOUPS} 2015)*, pages 53–67, 2015.

[19] S. Checkoway and H. Shacham. Iago attacks: why the system call api is a bad untrusted rpc interface. *ACM SIGARCH Computer Architecture News*, 41(1):253–264, 2013.

[20] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. H. Lai. Sgxpectre: Stealing intel secrets from sgx enclaves via speculative execution. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 142–157. IEEE, 2019.

[21] G. Chen, W. Wang, T. Chen, S. Chen, Y. Zhang, X. Wang, T.-H. Lai, and D. Lin. Racing in hyperspace: Closing hyper-threading side channels

on sgx with contrived data races. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 178–194. IEEE, 2018.

[22] S. Chen, X. Zhang, M. K. Reiter, and Y. Zhang. Detecting privileged side-channel attacks in shielded execution with déjá vu. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 7–18, 2017.

[23] Chromium. Google privacy sandbox. [Online]. Available: `https://www.chromium.org/Home/chromium-privacy/privacy-sandbox` (Retrieved, July 25, 2022).

[24] K. Conger and B. X. Chen. A change by apple is tormenting internet companies, especially meta, 2022. [ONLINE]. Available: `https://www.nytimes.com/2022/02/03/technology/apple-privacy-changes-meta.html` (Retrieved, July 25, 2022).

[25] W. De Groef, D. Devriese, N. Nikiforakis, and F. Piessens. Flowfox: a web browser with flexible and precise information flow control. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 748–759, 2012.

[26] W. De Groef, D. Devriese, N. Nikiforakis, and F. Piessens. Secure multi-execution of web scripts: Theory and practice. *Journal of Computer Security*, 22(4):469–509, 2014.

[27] R. Deepak and K. Nitish. Effect of disabling third-party cookies on publisher revenue, August 2019. [Online]. Available: `https://services.google.com/fh/files/misc/disabling_third-party_cookies_publisher_revenue.pdf` (Retrieved, July 25, 2022).

[28] E. Deng, H. Zhang, P. Wu, F. Guo, Z. Liu, H. Zhu, and Z. Cao. Pri-rtb:

Privacy-preserving real-time bidding for securing mobile advertisement in ubiquitous computing. *Information Sciences*, 504:354–371, 2019.

[29] D. Devriese and F. Piessens. Noninterference through secure multi-execution. In *2010 IEEE Symposium on Security and Privacy*, pages 109–124. IEEE, 2010.

[30] E. Elnikety, A. Mehta, A. Vahldiek-Oberwagner, D. Garg, and P. Druschel. Thoth: Comprehensive policy compliance in data retrieval systems. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 637–654, 2016.

[31] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS)*, 32(2):1–29, 2014.

[32] S. Englehardt, D. Reisman, C. Eubank, P. Zimmerman, J. Mayer, A. Narayanan, and E. W. Felten. Cookies that give you away: The surveillance implications of web tracking. In *Proceedings of the 24th International Conference on World Wide Web*, pages 289–299. International World Wide Web Conferences Steering Committee, 2015.

[33] D. Evtyushkin, R. Riley, N. C. Abu-Ghazaleh, ECE, and D. Ponomarev. Branchscope: A new side-channel attack on directional branch predictor. *ACM SIGPLAN Notices*, 53(2):693–707, 2018.

[34] G. Franken, T. Van Goethem, and W. Joosen. Who left open the cookie jar? a comprehensive evaluation of third-party cookie policies. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 151–168, 2018.

[35] Google. Fledge, May 2021. [ONLINE]. Available: https://developer.chrome.com/docs/privacy-sandbox/fledge/ (Retrieved, July 25, 2022).

[36] Google. Topic, January 2022. [ONLINE]. Available: https://developer.chrome.com/docs/privacy-sandbox/topics/ (Retrieved, July 25, 2022).

[37] Google. Content filtering, n.d. [ONLINE]. Available: https://support.google.com/adsense/answer/3011871?hl=en (Retrieved, July 25, 2022).

[38] Google. Infrastructure options for building advertising platforms, n.d. [Online]. Available: https://cloud.google.com/solutions/infrastructure-options-for-building-advertising-platforms (Retrieved, July 25, 2022).

[39] B. Gras, K. Razavi, H. Bos, and C. Giuffrida. Translation leak-aside buffer: Defeating cache side-channel protections with tlb attacks. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 955–972, 2018.

[40] M. Green, W. Ladd, and I. Miers. A protocol for privately reporting ad impressions at scale. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1591–1601. ACM, 2016.

[41] D. Gruss, J. Lettner, F. Schuster, O. Ohrimenko, I. Haller, and M. Costa. Strong and efficient cache side-channel protection using hardware transactional memory. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 217–233, 2017.

[42] S. Guha, B. Cheng, and P. Francis. Privad: Practical privacy in online advertising. In *USENIX conference on Networked systems design and implementation*, pages 169–182, 2011.

[43] M. Hardt and S. Nath. Privacy-aware personalization for mobile advertising. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 662–673. ACM, 2012.

[44] T. Hunt, Z. Zhu, Y. Xu, S. Peter, and E. Witchel. Ryoan: A distributed sandbox for untrusted computation on secret data. *ACM Transactions on Computer Systems (TOCS)*, 35(4):13, 2018.

[45] V. Kashyap, B. Wiedermann, and B. Hardekopf. Timing-and termination-sensitive secure information flow: Exploring a new approach. In *2011 IEEE Symposium on Security and Privacy*, pages 413–428. IEEE, 2011.

[46] T. Knauth, M. Steiner, S. Chakrabarti, L. Lei, C. Xing, and M. Vij. Integrating remote attestation with transport layer security. *arXiv preprint arXiv:1801.05863*, 2018.

[47] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, et al. Spectre attacks: Exploiting speculative execution. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1–19. IEEE, 2019.

[48] M. Krohn, A. Yip, M. Brodsky, N. Cliffer, M. F. Kaashoek, E. Kohler, and R. Morris. Information flow control for standard os abstractions. *ACM SIGOPS Operating Systems Review*, 41(6):321–334, 2007.

[49] S. Lee, M.-W. Shih, P. Gera, T. Kim, H. Kim, and M. Peinado. Inferring fine-grained control flow inside sgx enclaves with branch shadowing. In

26th USENIX Security Symposium (USENIX Security 17), pages 557–574, 2017.

[50] W. Li, H. Li, H. Chen, and Y. Xia. Adattester: Secure online mobile advertisement attestation using trustzone. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, pages 75–88, 2015.

[51] H. Liao, L. Peng, Z. Liu, and X. Shen. ipinyou global rtb bidding algorithm competition dataset. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*, pages 1–6. ACM, 2014.

[52] M. Mazmudar and I. Goldberg. Mitigator: Privacy policy compliance using trusted hardware. *Proceedings on Privacy Enhancing Technologies*, 1:18, 2020.

[53] W. Meng, R. Ding, S. P. Chung, S. Han, and W. Lee. The price of free: Privacy leakage in personalized mobile in-apps ads. In *NDSS*, 2016.

[54] A. Moghimi, G. Irazoqui, and T. Eisenbarth. Cachezoom: How sgx amplifies the power of cache attacks. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 69–90. Springer, 2017.

[55] D. Moghimi, J. Van Bulck, N. Heninger, F. Piessens, and B. Sunar. Copycat: Controlled instruction-level attacks on enclaves. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 469–486, 2020.

[56] M. Ngo, N. Bielova, C. Flanagan, T. Rezk, A. Russo, and T. Schmitz. A better facet of dynamic information flow control. In *Companion Proceedings of the The Web Conference 2018*, pages 731–739, 2018.

[57] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna. Cookieless monster: Exploring the ecosystem of web-based

device fingerprinting. In *2013 IEEE Symposium on Security and Privacy*, pages 541–555. IEEE, 2013.

[58] A. Nilsson, P. N. Bideh, and J. Brorsson. A survey of published attacks on intel sgx. *arXiv preprint arXiv:2006.13598*, 2020.

[59] O. Oleksenko, B. Trach, R. Krahn, M. Silberstein, and C. Fetzer. Varys: Protecting sgx enclaves from practical side-channel attacks. In *2018 Usenix Annual Technical Conference (USENIX ATC 18)*, pages 227–240, 2018.

[60] M. Orenbach, A. Baumann, and M. Silberstein. Autarky: Closing controlled channels with self-paging enclaves. In *Proceedings of the Fifteenth European Conference on Computer Systems*, pages 1–16, 2020.

[61] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard. Drama: Exploiting dram addressing for cross-cpu attacks. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 565–581, 2016.

[62] T. Pfeffer, T. Göthel, and S. Glesner. Efficient and precise information flow control for machine code through demand-driven secure multi-execution. In *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy*, pages 197–208, 2019.

[63] W. Rafnsson and A. Sabelfeld. Secure multi-execution: Fine-grained, declassification-aware, and transparent. *Journal of Computer Security*, 24(1):39–90, 2016.

[64] A. Reznichenko, S. Guha, and P. Francis. Auctions in do-not-track compliant internet advertising. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 667–676. ACM, 2011.

[65] A. Sabelfeld and A. C. Myers. Language-based information-flow security. *IEEE Journal on selected areas in communications*, 21(1):5–19, 2003.

[66] T. Schmitz, M. Algehed, C. Flanagan, and A. Russo. Faceted secure multi execution. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1617–1634, 2018.

[67] M. Schwarz, S. Weiser, D. Gruss, C. Maurice, and S. Mangard. Malware guard extension: Using sgx to conceal cache attacks. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 3–24. Springer, 2017.

[68] D. Sehr, R. Muth, C. Biffle, V. Khimenko, E. Pasko, K. Schimpf, B. Yee, and B. Chen. Adapting software fault isolation to contemporary cpu architectures. In *Proceedings of the 19th USENIX conference on Security*. USENIX Association, 2010.

[69] Y. Shen, H. Tian, Y. Chen, K. Chen, R. Wang, Y. Xu, Y. Xia, and S. Yan. Occlum: Secure and efficient multitasking inside a single enclave of intel sgx. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 955–970, 2020.

[70] M.-W. Shih, S. Lee, T. Kim, and M. Peinado. T-sgx: Eradicating controlled-channel attacks against enclave programs. In *NDSS*, 2017.

[71] S. Shinde, Z. L. Chua, V. Narayanan, and P. Saxena. Preventing your faults from telling your secrets: Defenses against pigeonhole attacks. *arXiv preprint arXiv:1506.04832*, 2015.

[72] smaato. Iab taxonomy, November 2021. [Online]. Available: `https:`

//developers.smaato.com/marketers/sdx-iab-taxonomy/ (Retrieved, July 25, 2022).

[73] R. Strackx and F. Piessens. The heisenberg defense: Proactively defending sgx enclaves against page-table-based side-channel attacks. *arXiv preprint arXiv:1712.08519*, 2017.

[74] G. Tan et al. *Principles and implementation techniques of software-based fault isolation*. Now Publishers, 2017.

[75] Technavio. Real time bidding market to grow by usd 16.52 bn, January 2022. [Online]. Available: https://www.prnewswire.com/news-releases/real-time-bidding-market-to-grow-by-usd-16-52-bn--abb-ltd-and-adobe-inc-among-key-vendors--technavio-301458196.html (Retrieved, July 25, 2022).

[76] V. Toubiana, A. Narayanan, D. Boneh, H. Nissenbaum, and S. Barocas. Adnostic: Privacy preserving targeted advertising. In *Proceedings Network and Distributed System Symposium*, 2010.

[77] C.-C. Tsai, D. E. Porter, and M. Vij. Graphene-sgx: A practical library os for unmodified applications on sgx. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 645–658, 2017.

[78] T. Tulabandhula, S. Vaya, and A. Dhar. Privacy-preserving targeted advertising. *CoRR*, abs/1710.03275, 2017.

[79] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx. Foreshadow: Extracting the keys to the intel sgx kingdom with transient out-of-order execution. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 991–1008, 2018.

[80] J. Van Bulck, D. Moghimi, M. Schwarz, M. Lippi, M. Minkin, D. Genkin, Y. Yarom, B. Sunar, D. Gruss, and F. Piessens. Lvi: Hijacking transient execution through microarchitectural load value injection. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 54–72. IEEE, 2020.

[81] S. Van Schaik, A. Milburn, S. Österlund, P. Frigo, G. Maisuradze, K. Razavi, H. Bos, and C. Giuffrida. Ridl: Rogue in-flight data load. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 88–105. IEEE, 2019.

[82] S. J. Vaughan-Nichols. Firefox blocks third-party web trackers by default, June 2019. [Online]. Available: `https://www.zdnet.com/google-amp/article/firefox-blocks-third-party-web-trackers-by-default/` (Retrieved, July 25, 2022).

[83] F. Wang, R. Ko, and J. Mickens. Riverbed: enforcing user-defined privacy constraints in distributed web services. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 615–630, 2019.

[84] Y. Xu, W. Cui, and M. Peinado. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *2015 IEEE Symposium on Security and Privacy*, pages 640–656. IEEE, 2015.

[85] N. Zeldovich, S. Boyd-Wickizer, E. Kohler, and D. Mazières. Making information flow explicit in histar. *Communications of the ACM*, 54(11):93–101, 2011.

[86] N. Zeldovich, S. Boyd-Wickizer, and D. Mazieres. Securing distributed systems with information flow control. In *NSDI*, volume 8, pages 293–308, 2008.

[87] W. Zhang, S. Yuan, and J. Wang. Optimal real-time bidding for display advertising. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1077–1086. ACM, 2014.

[88] Z. Zorz. Firefox 70 lets users track online trackers, October 2019. [Online]. Available: `https://www.helpnetsecurity.com/2019/10/23/firefox-see-online-trackers/` (Retrieved, July 25, 2022).

국문초록

온라인 광고에서는 사용자별로 맞춤화된 광고를 제공하기 위해 교차 사이트 추적 기법을 이용한다. 교차 사이트 추적은 사용자의 인터넷 활동 기록을 추적하는 방식으로, 이를 통해 사용자의 관심사를 알아낸다. 그러나 이러한 방식으로 수집된 데이터는 사용자의 민감 정보를 별도의 허가없이 수집하기 때문에 프라이버시 침해의 여지가 있다. 이 문제를 해결하기 위해 프라이버시를 보호하는 광고 시스템들이 제기되어 왔다. 그러나, 기존 연구에서는 사용자의 프라이버시를 보호하기 위해 광고의 유틸리티를 저해시킨다.

본 연구에서는, 프라이버시를 보호하는 광고 프레임워크인 Pave 를 제안한다. Pave 는 광고 회사들이 현재 광고의 유틸리티를 유지할 수 있고, Real-Time Bidding(RTB)과 같은 현재 광고 프로토콜을 지원한다. Pave 는 임의의 광고 프로그램에게 PaveBox 라는 실시간 모니터가 존재하는 격리된 실행 공간을 제공한다. Secure Multi-Execution(SME)으로부터 영감을 받은 PaveBox 는 모든 데이터 흐름을 가로채고 어떤 데이터 흐름이 사용자 데이터를 유출시킬 여지가 있다면 이를 금지시킨다. PaveBox 는 Intel SGX로부터 보호받기 때문에 프로그램의 무결성을 원격에서 검증할 수 있다. 본 연구에서는 형식화된 보안 모델을 수립하고, 이를 통해 프라이버시를 형식적으로 증명한다. 또한 프로토타입을 개발하여 실험을 통해 모델의 성능적 타당성을 보여준다.

주요어: 정보흐름제어, 안전한다중실행, 인텔 소프트웨어 가드 확장, 사용자 프라이버시, 프라이버시를 보호하는 광고 기술

학번: 2014-21784