ATTACKING SCHEDULE INDISTINGUISHABILITY IN REAL-TIME SYSTEMS

BY

DEBOPAM SANYAL

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois Urbana-Champaign, 2022

Urbana, Illinois

Adviser:

Adjunct Research Assistant Professor Sibin Mohan

ABSTRACT

Real-Time Systems (RTS) have gained prominence in new domains such as autonomous cars, drones, and the Internet-of-Things (IoT). RTS have stringent timing requirements for ensuring their correct operation. Such requirements make it necessary for systems to be deterministic at run-time. This determinism can be used against them as an attack surface, like scheduler side-channels. One way to reduce determinism in systems is by adding noise to system components in order to disrupt their deterministic behavior. Schedule indistinguishability, that is inspired by differential privacy, protects RTS by increasing the indistinguishability of the schedule.

It introduces the notion of ϵ-indistinguishability, that measures the probability of information leakage when system schedules are observed. Implemented in an ϵ-Scheduler, it is able to not only offer a higher degree of protection, but also do so with actual guarantees while still maintaining a high degree of performance. The efficacy of the ϵ-Scheduler is epitomized by its success in thwarting an identification attack on a real-time video streaming application. However, schedule indistinguishability can be compromised by side-channels, that leak critical information representative of the private timing data. This information can be combined with other observations made by an adversary to launch attacks on schedule indistinguishability.

The scope of attacks using side-channels is explored in this thesis. Two attacks, namely, a timing-based attack and a privacy budget attack are presented. While the timing-based attack takes advantage of the time taken to draw a sample from the noise distribution, the privacy budget attack uses the value of the protection duration. After detailing their threat models and showing their success in disrupting schedule indistinguishability, the thesis investigates their effectiveness in causing security breaches in RTS. The evaluations are carried out using the aforementioned identification attack on the video streaming application. Our results show that side-channel attacks can break schedule indistinguishability and, in turn, undermine the security of RTS against scheduler side-channels. The work can be divided into three key steps:

1. Ascertain the type of information that can be leaked by systems implementing the ϵ-Scheduler.
2. Determine the scope of side-channel attacks using the leaked information.
3. Analyze the relationship between schedule indistinguishability and security of RTS.

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my advisor Professor Sibin Mohan for guiding me throughout my research. I appreciate his help in all the areas of the project. I would like to extend my thanks to Chien-Ying Chen, for being a mentor whom I could turn to for help during various stages of the research. I also want to express my appreciation to Shahab Nikkhoo, who provided me with the resources essential for conducting the experiments.

TABLE OF CONTENTS

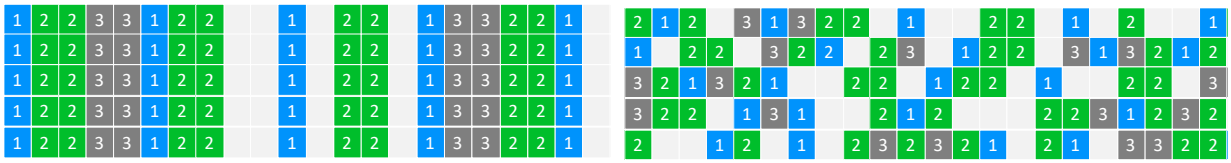
CHAPTER 1: INTRODUCTION

Real-time systems (RTS) are ubiquitous in modern technology. Often found in autonomous cars, robots, drones and medical devices, they play a crucial role in technological innovation. A real-time scheduler is vital for such systems to function because important tasks have to meet timing requirements. A periodic design and a guarantee of meeting deadlines of tasks helps RTS to maintain their utility.

Security is an important issue in RTS as any breach could cause disastrous effects in safety-critical applications that predominantly employ RTS [1, 2]. An ever-increasing number of attacks on RTS [3, 4, 5] show the need to focus more on their security. Since RTS have to meet time-sensitive requirements, their implementation often becomes predictable. Many tasks arrive periodically and have to be executed before their respective deadlines, hence giving rise to the predictability as shown in Figure 1.1(a).

While the predictability may help in maintaining real-time guarantees, it gives rise to timing side-channels that can leak some critical information about the system. In general, there are many side-channels like power consumption [7] and temperature [8], but we focus on a timing-based side-channel in this thesis called "scheduler side-channels". Figure 1.1(a) shows how across different hyper-periods the periodic tasks arrive and are executed at the same timestamps. While this kind of determinism helps in meeting deadlines, the schedule becomes very predictable. An adversary that observes the schedule for long enough can collect important information on the execution pattern of the system. This information, in turn, can aid the adversary in launching successful attacks [9]. Recently, there have been a large variety of attacks that target this vulnerability [10, 11]. There have also been proposed defense mechanisms to prevent adversaries from succeeding with these attacks [12, 13, 14]. The good mechanisms are the ones that do not significantly reduce the performance of RTS.

Schedule indistinguishability or ϵ-indistinguishability [6], defined in Chapter 3, is one such mechanism that reduces the success of attacks which seek to exploit scheduler side-channels, by randomizing the schedule instead of eliminating the presence of scheduler side-channels. Figure 1.1(b) shows the schedule of periodic tasks after applying randomization to their arrival times. Employing it not only gives protection to RTS, but the protection itself can be measured. This technique also achieves a trade-off between security and utility and provides a parameter used to tweak it, namely ϵ. Schedule indistinguishability heavily relies on mathematical principles from differential privacy [15, 16]. In differential privacy, data privacy is protected in the face of queries on databases by adding random noise. Schedule indistinguishability adds Laplace noise to a task's execution to break easily discernible



(a) Schedule before randomization (b) Randomized schedule

Figure 1.1: Reused from Chen *et al.* [6]. An example of the real-time schedule, consisting of 3 periodic real-time tasks, produced by a) a vanilla EDF scheduler and b) a randomized scheduler. It displays the schedules within a duration of 5 hyper-periods. In a) all hyper-periods have identical execution patterns due to the determinism in RTS. In b) hyper-periods are non-identical due to randomization used to disrupt the determinism in the schedule.

patterns in the task's schedule.

While differential privacy achieved using randomization techniques provides measurable privacy, the implementation of the randomization mechanisms inadvertently leaks some key information, thus potentially endangering privacy. Naturally, such leaks may be harmful for schedule indistinguishability as well. To fully understand the scope of schedule indistinguishability, it is important to check the effectiveness of attacks, especially ones that have been successful in compromising differential privacy. In this thesis, *two major attacks are devised, implemented and evaluated on a real-world application.* It is important to note that attacking schedule indistinguishability is not the same as attacking RTS using scheduler side-channels, however, there is a strong correlation between the two. Compromising schedule indistinguishability lays the foundation for scheduler side-channel based attacks on RTS.

1.1 RESEARCH GOAL

The work in this thesis starts off by making the following hypothesis:

There exist vulnerabilities in schedule indistinguishability applied to RTS due to side-channels leaking critical information about the private data.

Each chapter validates parts of this hypothesis and the conclusion states how it is finally proven. In order to validate the above hypothesis, we state the goal of the thesis as: *studying the scope of compromising schedule indistinguishability by exploiting its dependence on differential privacy, in order to attack RTS.* Analyzing the above objective will shed light on the viability of schedule indistinguishability in RTS. Some possible remedies against the proposed attacks are also suggested. We tackle questions like:

1. What type of information can be leaked from an implementation of schedule indistinguishability?

2. Is the information gained enough to mount successful attacks?

3. What is the relationship between attacking schedule indistinguishability and attacking RTS?

In this thesis, we consider attack scenarios mainly from a theoretical standpoint and show their potential on a video streaming application. We give less importance to their practical implementation. Hence, details like the type of adversary, the position of the adversary in the system, adversary's relationship to the system, *etc.*, are not researched exhaustively. We briefly present the background knowledge required and some important previous work related to the topic, before moving on to describe schedule indistinguishability. Then, we present the attack definitions and their experiment results. We finish with the conclusions drawn and mention some useful future directions of work.

CHAPTER 2: BACKGROUND AND RELATED WORK

Since the content of this thesis relies completely on real-time systems (RTS), it is imperative to understand how they work and what their key properties are. Because the notion of schedule indistinguishability is attacked in the thesis, it is vital to know what it is, why it is required and the techniques used to achieve it. This section further provides a brief overview of differential privacy as it is the basic mathematical tool on which schedule indistinguishability lies. We also mention important related work that has helped inform our research and findings.

2.1 SECURITY IN REAL-TIME SYSTEMS

As discussed in the introduction, RTS with real-time constraints are predictable owing to determinism in their run-time timing behavior. The predictability stems from the periodic nature of some critical software components, like tasks being constrained by predefined inter-arrival times. Other important features in RTS are deadlines, worst-case execution times (WCETs) and real-time scheduling algorithms like earliest deadline first (EDF) [17], rate monotonic [17], and first-come-first-serve [18, 19]. To ensure such systems uphold real-time properties such as deadline guarantees, their schedules often form a repeatable pattern that is both deterministic and predictable. Important related work that focus on securing RTS have traditionally been tailored to detect inconsistencies and reduce effectiveness of general attacks [20, 21, 22, 23, 24, 25]. These general attacks, however, do not include attacks exploiting scheduler side-channels.

While the repeated schedule patterns are crucial for the system to deliver guarantees, the predictability generated in execution gives rise to side-channels that can be exploited by adversaries to mount attacks. The limited uncertainty due to the repeatability of schedules allows the adversary to have a high chance of success on forecasting future arrival times of jobs in the system. Over the past few years, numerous side-channels have been reported in RTS [7, 8, 26] and there has been some work studying the existence of side-channels and covert-channels due to determinism in RTS [11, 27, 28, 29]. Attacks, too, can be of various types like timing side-channel attacks, or other covert-channel attacks, where two processes covertly communicate using the scheduler [30]. As discussed before, we are dealing with scheduler side-channels in this thesis which are a type of timing side-channel. These side-channels leak critical timing behavior of task schedules. Chen *et al.* [10] were the first to introduce the notion of scheduler side-channels in fixed-priority preemptive RTS.

2.1.1 ScheduLeak

ScheduLeak [10] algorithms can successfully extract timing information of important real-time tasks from simply observing a task schedule at run-time. The main motive of this class of attacks is to collect timing information of vital tasks. This includes: (i) when they occur, (ii) how often they occur and (iii) when (and if) they will occur again. Precisely, the presence of scheduler-based side-channels leak timing information, where an unprivileged, low-priority task learns the timing behavior of other periodic (victim) tasks. ScheduLeak is carried out by observing the execution times using a timer. The information gained by the adversary, via the low-priority task, allows it to reconstruct the initial offsets and predict the future arrival times of victim tasks. And at these arrival times, it can launch attacks against the victim tasks. Without this information, it would have to launch its attacks at random points in time, thus making the attacks less successful. The timing information leaked has the potential to help launch other attacks dangerous to a real-time system. Adversaries can determine periods of real-time tasks with high precision using methods like Discrete Fourier Transform. The knowledge of periods allows adversaries to launch other attacks. Moreover, it is easier to create a user space application that obtains information about victim tasks instead of targeting the scheduler of the system to extract the required information.

2.2 SCHEDULE OBFUSCATION

There has been work [14] associated with randomization of scheduling in fixed-priority RTS that aims to obfuscate the task schedules in order to prevent scheduler side-channels. Another method [13] developed a randomization paradigm to reduce determinism in time-triggered systems. Nasri *et al.* [31] conducted a comprehensive study on the schedule randomization approach and argued that such techniques can expose the fixed-priority preemptive RTS to more risks. Although the above methods are centered on the problem of scheduler side-channels, they do not provide analytical guarantees on the protection against scheduler side-channels. Additionally, the above methods target hard RTS that are highly constrained by strict real-time guarantees and hence, their effectiveness is limited by the real-time nature. In contrast, we focus on a more realistic real-time system model that has flexible and more tolerable timing requirements. This enables space for exploring more aggressive defense strategies to achieve higher (and analyzable) protection against the threats imposed by scheduler side-channels.

2.3 DIFFERENTIAL PRIVACY

Differential Privacy was introduced by Dwork [15, 16] in the context of statistical queries on databases and has grown into a widely used technique to ensure data privacy [15, 32]. It guarantees that a malicious querier (adversary) cannot reason with high confidence about the presence of a particular data entry by just looking at the outputs of queries returned by differentially private mechanisms. Moreover, such protection is quantifiable based on the randomization mechanism used. Note that while the high-level goals are similar, leakage of private data is the typical use case for differential privacy as opposed to deterring scheduler side-channel attacks, that is the typical use case for schedule indistinguishability.

In our context, there are task and job level indistinguishabilities that define the probability of distinguishing the execution states of one task/job from another in task schedules. Roughly speaking, a low indistinguishability enables an adversary to identify a task's execution from an observed schedule with high confidence and hence makes the system prone to compromises via scheduler side-channels. To address such a problem, we have an ϵ-Scheduler [33] that offers "ϵ-indistinguishability" at job level and/or task level, subject to system constraints as well as the system designer's security goal. This is achieved by embedding a randomized scheduling mechanism for adding noise to the inter-arrival times for each job at every scheduling point to abate the predictability and determinism.

2.3.1 Laplace Mechanism

There exist many types of distributions that are used for addressing information leakage issues (*e.g.*, Uniform distribution [34], Gaussian distribution [35, 36] and Laplace distribution [16].) The Uniform distribution has been used in the KeyDrown [34] work to prevent information leakage via keystrokes. In our context, while using the Uniform distribution would spread the inter-arrival times more evenly (*i.e.*, better security), it would cause more deadline misses and a serious degradation of the real-time performance. The Laplace distribution has been used in classic differential privacy problems for generating random noise to achieve desired privacy protections [16]. Conventionally, the Laplace distribution has a probability density function defined as:

$$\text{Lap}(x \mid \mu, b) = \frac{1}{2b}\exp(-\frac{|x-\mu|}{b}) \tag{2.1}$$

where μ is a location parameter and $b > 0$ is a scale parameter. The distribution has a mean equal to μ and a variance equal to $2b^2$. We use the Laplace distribution to generate random-

ized inter-arrival times for each job at run-time. In contrast to the Uniform distribution, the Laplace distribution, while statistically vulnerable over time, can generate an average task period that's close to the designer's desired value thus closely matching required real-time guarantees. Furthermore, using the Laplace distribution allows us to reuse existing mathematical and algorithmic components with the theoretical foundations from the differential privacy domain.

CHAPTER 3: SCHEDULE INDISTINGUISHABILITY

At its core, schedule indistinguishability seeks to break the determinism in schedules of RTS. Precisely, it utilizes randomization techniques to obfuscate task schedules, thereby disrupting the predictability in schedules of RTS. This is achieved by adding a sufficiently large (controlled) noise to the task schedules in order to break their deterministic execution patterns. Not only does it introduce diversity into the schedules of such systems, but also provides a scope for analyzable security guarantees. The goal is to offer protection against scheduler side-channels in RTS while still maintaining good performance and quality-of-service (QoS) requirements. Schedule indistinguishability captures the difficulty of identifying information about individual tasks in a task schedule. Chen *et al.* [6] were the first to define schedule indistinguishability. This chapter contains the salient points that will be used in the Chapters 4 and 5 to define attacks on schedule indistinguishability.

3.1 PRELIMINARIES

We denote the Laplace distribution with location μ and scale b by $Lap(\mu, b)$ and we write $Lap(b)$ when $\mu = 0$. We assume that a unit of time equals a timer tick governed by the operating system and the corresponding tick count is an integer. That is, all system and real-time task parameters are multiples of a time tick. Throughout this thesis, we consider a single processor, preemptive real-time system in which some deadline misses are tolerable [37, 38]. The system contains a set of N real-time tasks, $\Gamma = \{\tau_i \mid i \in [N]\}$, scheduled by a dynamic-priority scheduler like EDF [17]. We assume the real-time tasks are independent of each other. We model a real-time task τ_i by a tuple $(\mathcal{T}_i, \mathcal{D}_i, C_i, \eta_i)$ where $\mathcal{T}_i = \{T_{i,k} \mid k \in \mathbb{N}\}$ is a set of admissible periods, $\mathcal{D}_i = \{D_{i,k} \mid k \in \mathbb{N}\}$ is a set of implicit, relative deadlines ($D_{i,k} = T_{i,k}, \forall k \in \mathbb{N}$), C_i is the worst-case execution time (WCET) and η_i is a task inter-arrival time function. For a periodic task, $\mathcal{T}_i = \{T_{i,1}\}$, so we sometimes use T_i to denote such a fixed period for simplicity. The inter-arrival time function is illustrated in Figure 3.1. It is defined as follows:

$$\eta_i : \mathbb{N} \rightarrow \mathcal{T}_i \tag{3.1}$$

where τ_i is the task and $\eta_i(j)$ is the task's inter-arrival time at the j^{th} instance. The resulting inter-arrival time is one of the values in the task's inter-arrival time set, $\eta_i(j) \in \mathcal{T}_i$. Note that a strict periodic task always gets a fixed output from its inter-arrival time function, $\eta_i(j) = T_{i,1}, \forall j \in \mathbb{N}$. So, when the j^{th} instance of task τ_i arrives, the scheduler computes its

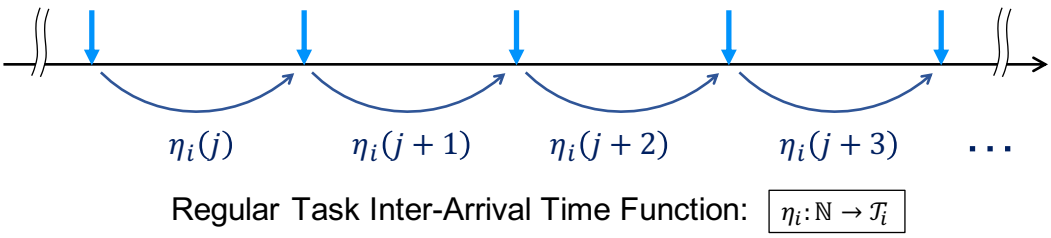


Figure 3.1: Reused from Chen *et al.* [6]. Illustration of the task execution model. Arrows represent the scheduled arrival time instants. The distance between two adjacent arrival times of a task is modeled by a task-specific function η_i.

period from $\eta_i(j)$ and configures its deadline as well as the next arrival time accordingly.

We are mainly concerned about scheduler side-channels that are exposed by the deterministic nature of RTS as introduced in Chapter 2. Hence, we assume that an adversary observes the system schedule via some existing side-channel [7, 8, 10, 26] and that the adversary does not have access to the scheduler. Note that some existing attacks have demonstrated that precise timing information of target tasks can be deduced from the deterministic real-time schedules at run-time [10, 39]. Such information can then be exploited to recreate a targeted task's execution state and then launch further, more critical, attacks on the system with higher precision. As demonstrated in ScheduLeak [10], an attacker can place precise prime and probe operations before and after a target task and successfully learn its execution behavior. These types of attacks rely on the fact that periodicity exists in the real-time tasks that are being targeted. Schedule indistinguishability aims to eliminate such scheduler side-channels by obscuring the task periodicity in the schedule. Schedule indistinguishability can be categorized into: job-level indistinguishability, where indistinguishability between jobs of the same task in the task set is considered and task-level indistinguishability, where indistinguishability between tasks of the task set is considered.

Now, we recap some definitions crucial for understanding the ϵ-scheduler and the attacks detailed in Chapters 4 and 5. We stick to the job-level indistinguishability definitions as all the attacks in this thesis are based on them. The inter-arrival time randomized mechanism, denoted by $\mathcal{R}(\cdot)$, is attached to the scheduler to add random noise. It is defined as:

$$\mathcal{R}(\tau_i, j) = \lfloor \eta_i(j) + Y \rceil \tag{3.2}$$

where $\tau_i \in \Gamma, j \in \mathbb{N}$ represent the j^{th} inter-arrival time of the task τ_i. Y is a random noise value drawn from some distribution centered at 0. This is the same as drawing a random value from some distribution centered at $\eta_i(j)$ – which is how the ϵ-Scheduler functions. The outcome is rounded to the nearest integer. To analyze job-level indistinguishability with the

mechanism defined in Equation 3.2, we use a concept that's similar to the notion of differential privacy [15, 16]. An inter-arrival time randomized mechanism is ϵ-indistinguishable if:

$$\Pr[\mathcal{R}(\tau, j) \in \mathcal{S}] \leq e^{\epsilon} \Pr[\mathcal{R}(\tau', j') \in \mathcal{S}] \tag{3.3}$$

for all $\tau, \tau' \in \Gamma,\ j, j' \in \mathbb{N}$ and $\mathcal{S} \subseteq \text{Range}(\mathcal{R})$. This means $\mathcal{R}(\cdot)$ enables inter-arrival time indistinguishability for a single job instance if Equation 3.3 is satisfied. To determine the degree of noise to be added to make two inter-arrival times indistinguishable, we use inter-arrival time sensitivity, that reflects the sensitivity of the function $\eta_\tau(\cdot)$. For job-level indistinguishability, denoted by $\Delta\eta_\tau$, for a given task τ, it is defined as:

$$\Delta\eta_\tau =: \max_{\substack{j, j' \in \mathbb{N} \\ j \neq j'}} |\eta_\tau(j) - \eta_\tau(j')| \tag{3.4}$$

Then, the use of the Laplace distribution $\text{Lap}(\eta_\tau, \frac{\Delta\eta}{\epsilon})$ for generating the randomized inter-arrival times preserves ϵ-indistinguishability from Equation 3.3 for a single job instance. However, an attacker observes a longer sequence from the schedule. As a noise draw occurs for every job instance, based on the theorem of Sequential Composition [40], the privacy degradation is cumulative as the number of draws increases. We measure the duration in the number of job instances, *i.e.*, the number of noise draws for the corresponding inter-arrival times. Since the Laplace randomized mechanism $\mathcal{R}(\cdot)$ with the scale $\frac{J\Delta\eta}{\epsilon}$ is ϵ-indistinguishable up to J job instances, it can be used to determine the scale of the noise required for preserving ϵ-indistinguishability up to J job instances.

While the introduced Laplace randomized mechanism offers ϵ-indistinguishability, the unbounded output domain for the randomized inter-arrival times makes it infeasible to adopt in RTS. To address this problem, the randomized inter-arrival time is drawn from a Laplace distribution that is bounded by a given range. Bounding is achieved by continually sampling independently from the distribution until a value within the specified range is returned. Let $[T_i^{\perp}, T_i^{\top}]$ be the feasible inter-arrival time range for a given task τ_i, the bounded inter-arrival time Laplace randomized mechanism is defined as:

$$\widetilde{\mathcal{R}}(\tau_i, j) = \widetilde{L}(\eta_i(j), \frac{2J_i\Delta\eta_i}{\epsilon_i}, T_i^{\perp}, T_i^{\top}) \tag{3.5}$$

where $\widetilde{L}(\cdot)$ is the bounded Laplace distribution of which the drawn values are bounded in the range $[T_i^{\perp}, T_i^{\top}]$ based on a pure Laplace distribution $\text{Lap}(\eta_i(j), \frac{2J_i\Delta\eta_i}{\epsilon_i})$.

3.2 ϵ-SCHEDULER

The basic task of the ϵ-scheduler is to implement schedule indistinguishability in real-time Linux based on bounded Laplace distributions. On each job's arrival (the beginning of a new instance), the ϵ-Scheduler uses $\widetilde{\mathcal{R}}(\cdot)$ for generating the task's next arrival time. The basic task model in Section 3.1 is extended: a task τ_i is characterized by $(\mathcal{T}_i, \mathcal{D}_i, C_i, \eta_i, T_i^{\perp}, T_i^{\top}, \Delta\eta_i, J_i, \epsilon_i)$ where $[T_i^{\perp}, T_i^{\top}]$ is a range of tolerable periods, $\Delta\eta_i \geq 0$ is the inter-arrival time sensitivity parameter, J_i is the task's effective protection duration, and $\epsilon_i > 0$ is the indistinguishability scale parameter. At each new job arrival, the ϵ-Scheduler invokes $\widetilde{\mathcal{R}}(\tau_i, j) = \widetilde{L}(\eta_i(j), \frac{2J_i\Delta\eta_i}{\epsilon_i}, T_i^{\perp}, T_i^{\top})$ to determine the next job's randomized arrival time point. In this extended task model, the parameters $\mathcal{T}_i, \mathcal{D}_i, C_i, \eta_i, T_i^{\perp}$ and $T_i^{\top}$ are obtained from the system dynamics. The additional parameters $\Delta\eta_i$, J_i and ϵ_i are to be given by the system designer. If we intend to achieve job-level indistinguishability for a given task τ_i, the value of $\Delta\eta_i$ is determined solely by the task's set of periods, $\mathcal{T}_i$. In this case, each task's sensitivity is independent from the others. Finally, it is up to the system designer to decide, taking potential performance degradation into account, which type of indistinguishability should be achieved.

Using the bounded Laplace mechanism, $\widetilde{\mathcal{R}}(\cdot)$, an ϵ-Scheduler is able to preserve ϵ_i- indistinguishability up to J_i job instances for a given task. The more noise samples collected, the more likely is an attacker to be able to reconstruct the distribution. Therefore, ϵ_i-indistinguishability can't be guaranteed for an infinite time. A feasible solution is performing periodic security checks to detect possible intrusions and anomalies [41]. With such a scheme, the distance between two security checks can be used as a reference to compute the protection duration parameter J_i. Another feasible scheme is the restart-based mechanism [42, 43] that enforces a reboot every once in a while. In such a case, the maximum time to reset the system can be used to compute J_i. When job-level indistinguishability is considered, the protection duration in time, λ, is defined as:

$$J_i = \left\lceil \frac{\lambda}{\min(\mathcal{T}_i)} \right\rceil \tag{3.6}$$

Equation 3.6 offers ϵ_i-indistinguishability to τ_i within λ time, where λ is task-specific. The next variable to be specified is ϵ_i. Choosing a smaller ϵ_i value provides better indistinguishability by generating randomized inter-arrival times with larger noise scale. However, a large noise scale may sometimes be impractical for real-time applications. Figure 3.2(a) suggests that an ϵ_i value above an order of magnitude of J_i can be practical for most RTS. Nevertheless, a suitable value for ϵ_i is highly system-dependent. Each task in a task set can have an

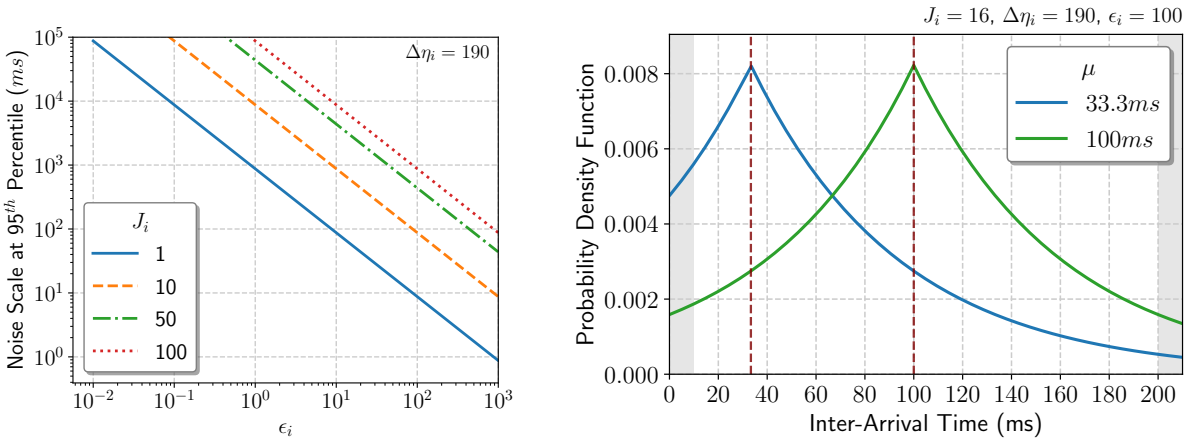
independent ϵ value for job-level indistinguishability.

3.2.1 Implementing the Laplace Distribution

ϵ-Scheduler requires the generation of random numbers based on Laplace distribution for obtaining randomized inter-arrival times. However, the Linux kernel code is self-contained and thus a random number generator that's based on Laplace distribution is not natively supported. While it is possible to build such a generator out of the existing random number generation function `get_random_bytes`(), the required computations will be costly. Considering that the task set parameters are fixed at the design stage, the Laplace distributions needed by each task are fixed and known as well. Therefore, rather than building a common Laplace distribution-based random number generator, each required Laplace distribution's percent point function (PPF) is converted into an array and stored in the kernel [6]. Then, a Laplace distribution-based random number can be drawn by randomly picking (via `get_random_bytes`()) a number from the array that's associated with the desired Laplace distribution. We call this default method of calculating the Laplace distribution in ϵ-Scheduler as the "look-up sampling" method as we are essentially looking up values from an array based on the sampling inherent in `get_random_bytes`(). In Chapter 4, we contrast this method to a geometric sampling method [44] used to calculate the Laplace noise. Other details of the implementation of ϵ-Scheduler can be found in Section 6 of Chen *et al.* [6].

3.3 VIDEO STREAMING AND IDENTIFICATION ATTACK

Schedule indistinguishability is evaluated using a real-world video streaming application that sends data across large geographic distances [6]. Generally, it is not always easy to use the predictable system behavior to learn the run-time information, but a number of assumptions made in the video streaming application, mentioned below, makes it easier to extract precise timing behavior from observing the deterministic schedules. We built a video streaming application using Dynamic Adaptive Streaming over HTTP (DASH) as the video streaming standard and flask [45] for our web application. Our goal was to show that the ϵ-Scheduler is useful in negating traffic-based attacks (types of data leakage attacks [46]) on such video streaming applications without significantly affecting the performance of the application itself. The video stream is hosted by a server and the client is the receiver of the video stream that is transmitted via the application over the internet over a distance of 1800 miles. Our attacker is placed in between the server and the client, so that eavesdropping on the network traffic can be easily carried out. The performance of the application is measured



(a) Noise scale of Lap$(0, \frac{2J_i\Delta\eta_i}{\epsilon_i})$

(b) Probability density

Figure 3.2: Reused from Chen *et al.* [6]. (a) The noise scale of Lap$(0, \frac{2J_i\Delta\eta_i}{\epsilon_i})$ at 95^{th} percentile with $\Delta\eta_i = 190ms$ and varying ϵ_i and J_i. Both axes are displayed in a base 10 logarithmic scale. (b) Probability density of the randomized inter-arrival times for the task τ_i with $\mathcal{T}_i = \{33.33ms, 100ms\}$. The blue and green lines show the distribution when the desired period is at $33.33ms$ and $100ms$ respectively. In this case, ϵ-Scheduler offers job-level ϵ_i-indistinguishability for τ_i with $\epsilon_i = 100$, $\Delta\eta_i = 190$ and $J_i = 16$.

using the frames per second (FPS) of the video. Ideally, the FPS of the video at the client's end is similar to the FPS of the video sent from the server. It is important to note that the ϵ-Scheduler only randomizes the arrival time of video frames to the client and does not change the content of the video. An attack can be devised on such applications by exploiting some key properties of DASH video streaming.

The default values for experiments with ϵ-Scheduler are: $\Delta\eta = 190ms$, $J = 16$ with the desired protection duration to be $\lambda = 500ms$ for the video streaming task running at $30Hz$. Our evaluation verifies whether the video identification attack in the case of the ϵ-Scheduler shows results that are random at best. In our setup, we use five videos with varying content, frame rates and resolutions. We consider a total of five streaming scenarios, each scenario being the event when only one of the five videos is being streamed via our application, *i.e.*, only video x is being streamed, where $x \in \{1, 2, ..., 5\}$. Eavesdropping is done for 30 seconds with 2 seconds as the segment length and the corresponding traffic patterns are captured using Wireshark. For repeatability, this is done fifteen times for each scheduler (Vanilla EDF, ϵ-Scheduler with $\epsilon = 10$ and $\epsilon = 10^3$), resulting in 45 traffic pattern samples for each scenario.

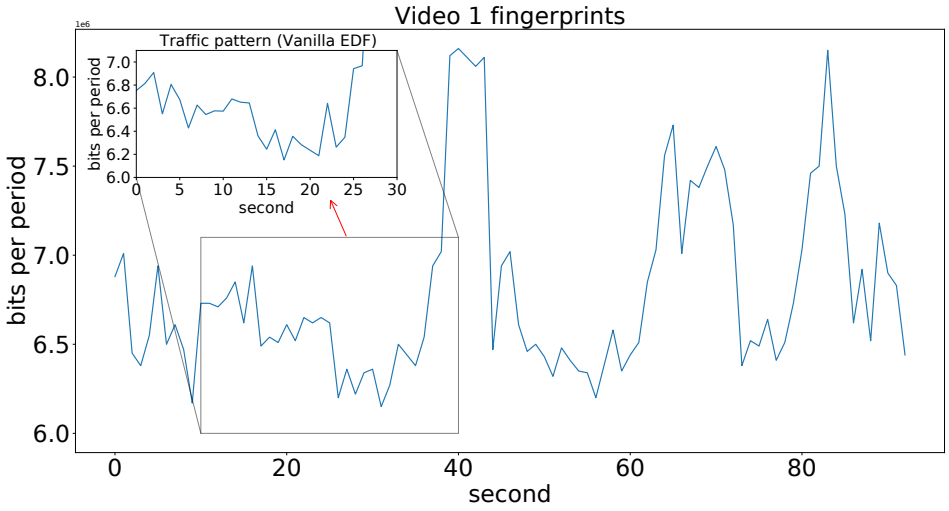


Figure 3.3: The similarity between the eavesdropped traffic pattern with Vanilla EDF when video 1 was being streamed and video 1's fingerprints. The similarity distance is calculated using temporal sequence analysis.

3.3.1 The Matching Method

While streaming with DASH, each video segment is a certain segment length and quality level. This type of mechanism results in a distinct traffic pattern due to the segment-based transmission. This key property can be used to identify videos while streaming. By eavesdropping on the network traffic during video streaming, attackers can recognize certain patterns in the traffic. Video fingerprints can be built on the other hand, using the pre-recorded video files. Hence, attackers can utilize the fingerprints and the observed traffic pattern to identify which video was being streamed during eavesdropping. The idea is to merely compare the extracted video fingerprints with the observed traffic pattern of the video stream to deduce an individual video.

A matching method is necessary for effective outcomes post-comparison. In this way, the eavesdropped traffic and the video fingerprints provide seamless video identification. The length of the eavesdropped sample is an obstacle, however. It is time-consuming to eavesdrop on the entire video. Also most times, the host doesn't play the entire video, so only a part of the video is present in the eavesdropped traffic. The systematic steps are: extracting the video fingerprints from pre-recorded videos, obtaining the eavesdropped traffic pattern, calculating a similarity estimate between the traffic pattern and the fingerprints using temporal sequence analysis and finally, identifying the video using this similarity estimate. Specifically, we use the fingerprint-based video identification attack detailed in Gu *et*

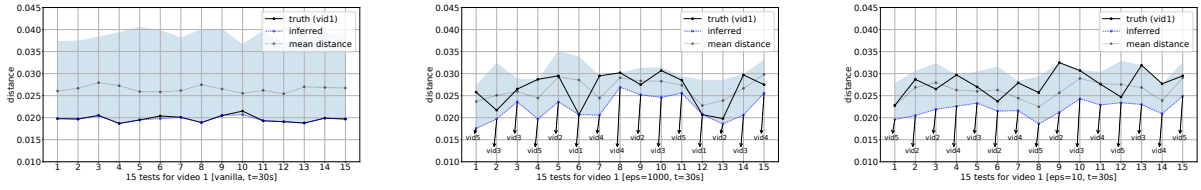
Table 3.1: FPS Observed by the Client (Video 1)

Scheduler	Max	Mean	Min	Std	CV
Vanilla EDF	32.00	29.90	28.00	0.68	0.02
ϵ-Scheduler ($\epsilon = 10^3$)	28.00	25.87	22.00	1.71	0.07
ϵ-Scheduler ($\epsilon = 10$)	14.00	9.23	6.00	2.07	0.22

al. [46], adapting it to the video streaming application that we designed.

The fingerprints that we obtain follow the segmentation rules of DASH. Initially, we calculate the data per second in bits per second (bps) for each video in a differential manner in order to eliminate the impact of different quality levels on the fingerprints. This sequence of fingerprints, which correspond to data per second, need to be aggregated into segments before the matching step because DASH transmits video data in segments. Each segment covers a certain number of seconds; segment length is usually kept constant. Again a differential strategy is used to collect the data per segment and the resulting set is the set of video fingerprints available to the attacker. For each video in the dataset, a set of video fingerprints is calculated. Next, the attacker eavesdrops on the network traffic during the transmission of the video to obtain traffic traces. Assuming that there are no other processes that require a large bandwidth, the attacker aggregates the obtained network traffic (in bps) into data per segment in a differential manner. The objective is to find a maximum match between the sequence of traffic traces and the sequence of individual video fingerprints. The video corresponding to the maximum match is identified to be the video that was being streamed during the eavesdropping period.

The similarity measurement is a method to find out which set of video fingerprints is likely to produce the extracted traffic pattern. Since the assumption is that only one video is streamed at a time, measuring the similarity of the pattern to each set of fingerprints will reveal the closest match. We treat this as a time series matching problem. Two important considerations before solving the problem have to be taken into account: eavesdropping can be short and the eavesdropped period may correspond to only a portion of the entire video. Hence, after normalizing the sequences using a sigmoid function, a method called partial dynamic time warping (p-DTW) is used. p-DTW finds the best local alignment between the two sequences, *i.e.*, it minimizes the distance between the traffic pattern and any proper subsequence of the fingerprints. The sub-sequence that results in this minimum is selected for calculating the final similarity between the fingerprint and the traffic traces. The similarity is quantified using the minimum distance (cost) of aligning the two sequences. Whichever set of video fingerprints renders the minimum distance is the identified video in our attack.



(a) Vanilla EDF (b) ϵ-Scheduler ($\epsilon = 10^3$) (c) ϵ-Scheduler ($\epsilon = 10$)

Figure 3.4: The similarity distance measures for 15 traffic samples. We shaded the region between the maximum and the minimum distances. The mean distances are shown in gray dotted lines. Clearly, the similarity distances between the traffic patterns and video 1's fingerprints are rarely the minimum across the 15 tests with ϵ-Scheduler (2 out of 15 with $\epsilon = 10^3$ and 0 out of 15 with $\epsilon = 10$), while they are the minimum with Vanilla EDF in 13 out of 15 tests. This shows that the random noise added to the traffic patterns under ϵ-Scheduler reduces the effectiveness of the traffic-based video identification attack.

3.3.2 Experiment Results Summary

For identification purposes, we calculate the distance metric *dist*, which is representative of the similarity between a given traffic pattern and a video fingerprint. Hence, given a traffic pattern and a dataset containing n videos, there are n distances generated ($n = 5$ in our case). The smaller the value of *dist*, the greater the probability for the traffic pattern matching the video fingerprint, which is equivalent to the probability of the corresponding video being streamed during the eavesdropping. In order to compensate for our relatively short eavesdropping time, instead of setting thresholds on distances to identify the target video [46], we simply identify the target video as being the one that had the minimum distance out of the five calculated distances. Figure 3.4 shows the similarity distances for the traffic samples collected when video 1 was being streamed. The results obtained when the other videos (2,3,4 and 5) were being streamed, closely match that of video 1.

Table 3.1 shows the FPS statistics observed at the client's end over a duration of 30 seconds for video 1 in which an FPS data point is computed using the number of frame packets received from the Internet per 0.5 seconds. The FPS of video 1 sent from the server is 30. The Coefficient of Variation (CV) value represents relative variability of the FPS in each scheduler configuration. It reveals that FPS in the case of Vanilla EDF has the smallest variability as there is no randomization while it shows the largest variability in the case of ϵ-Scheduler with $\epsilon = 10$. In the case of ϵ-Scheduler with $\epsilon = 10^3$, it has a reasonably small CV value and slightly decreased mean FPS with good protection against the eavesdropping attack that's comparable to $\epsilon = 10$ (see Figure 3.4). As a result, it shows that a balance between performance (FPS) and security can be reached by using ϵ-Scheduler with $\epsilon = 10^3$ in this experiment.

CHAPTER 4: A TIMING-BASED ATTACK

This attack exploits a common implementation flaw in the noise generation process used in differential privacy. Since the ϵ-Scheduler uses a noise generation scheme very similar to the ones found in differential privacy, this attack can potentially compromise the guaranteed indistinguishability. A discrete Laplace mechanism works with the ϵ-Scheduler that suffers from a novel timing side-channel [47, 48]. An adversary can observe the time taken to draw discrete Laplace noise. Such time information can be used to predict the magnitude of the noise being added or subtracted by the inter-arrival time randomized mechanism that can then be employed to discover other sensitive information about tasks. The proposed attack exploits this side channel to gauge information about the noise being added to jobs in a task. The attack model is kept general enough so that it can be applied to any application using RTS. We check the validity of this attack on our video streaming application and go on to suggest possible remedies.

4.1 DEFINITION AND SCOPE

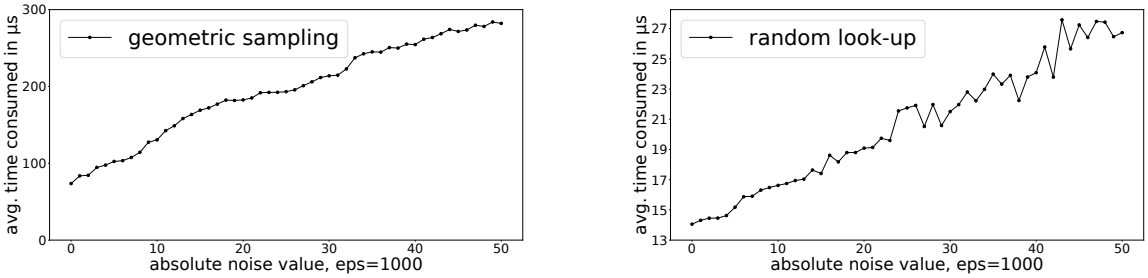
While discrete versions of Laplace mechanisms [44, 49] defend against floating-point attacks [50], the sampling strategies employed in these approaches open up another timing side-channel. Here the timing side-channel is the time taken to produce a sample from the Laplace distribution. Our experiments show that this information enables the adversary to predict the noise magnitude with greater certainty. The attack works with the ϵ-Scheduler in an offline setting. For experiment purposes, we test with both the default look-up style sampling method used by the ϵ-Scheduler and a geometric sampling method [44, 51]. The look-up style sampling method is described in Chapter 3. The other method tries to simulate geometric sampling, where sampling is done until a value within the range is returned. The number of trials is related to the magnitude of the noise returned. Hence, timing the sampler can indicate this number thus leaking the magnitude of the noise. The key difference between this side-channel and general timing side-channels is that here we are concerned with timing the sampling algorithm instead of timing the randomization mechanism or the timing in the schedules. It is important to keep in mind that avoiding discrete samplers would result in the existence of floating-point side-channels [50].

Discrete samplers aim to retain privacy properties of continuous representations, while not allowing information to be leaked via floating-point side-channels. In this attack, the adversary times the noise sampling algorithm. The time calculated helps to determine the

magnitude of the unknown noise. A naive discrete sampler suffers from this type of a timing side-channel. For our experiments, we have two types of discrete Laplace samplers. The first one is the default sampling method in the ϵ-Scheduler [33], where the discrete values are already available, but which value to choose and the sign are randomly determined using the `get_random_bytes`() function that exists in the Linux kernel. The second method uses the same principles of the default ϵ-Scheduler, except it has a geometric sampling based Laplace noise generation technique [51]. Here the sampler uses a geometric distribution where the magnitude of the noise drawn is preserved by corresponding samples. It is the time taken to return a value by this geometric sampler that reveals the magnitude of the noise. There is another method that approximates the discrete Laplace distribution [49], but it also suffers from the timing side-channel, where the time taken to generate larger values is longer, due to the fact that larger magnitudes have smaller probabilities of being selected.

The geometric sampling method uses Bernoulli samples for the geometric distribution [44, 51], where we measure the number of Bernoulli trials required for success. Basically, the number of failures is counted and the sampling is done until the first success. The count of failures, n, is treated as a sample of the geometric distribution. This results in a linear correlation between the number of times the Bernoulli sampler executes and the magnitude of the sample n. Since the number of trials cannot be observed by the adversary, *the running time serves as the side-channel*, and is correlated to the unknown value that has just been drawn. Finally, a linear transformation of the drawn value gives the required Laplace distribution value. This way we get to preserve not only the magnitude, but also the exact discrete distribution, by employing rejection sampling.

The timing side-channel does not originate from the rejection sampling, but instead from the stage where a value is drawn from the geometric distribution. Thus, the time taken by the discrete Laplace sampling to run has a correlation with the magnitude of the value that has been returned, which is nothing but the noise introduced for schedule indistinguishability. Our experiments show that this is indeed the case (see Figure 4.1). Even though the correlation is positive, it is only roughly linear. We suspect that baseline experiments for a much larger range of discrete noise values with many repetitions will more closely depict the linear nature of the correlation. From Table 4.1, we see that the accuracy of the attack is consistently less with the lookup method than with the geometric method. A possible reason why the lookup method is better protected is because it does not follow the linear positive correlation between the magnitude and the time taken as well as the geometric sampling technique does, especially for large magnitudes as seen in Figure 4.1.



(a) Geometric Sampling

(b) Look-up based sampling

Figure 4.1: Average time to generate absolute noise, in μs, using two implementations in ϵ-Scheduler: (a) Default random lookup from array (avg. over 10,000 trials). (b) Geometric sampling based Laplace implementation (avg. over 10,000 trials). The plots show a roughly linear relationship between the noise magnitude and the time it takes to generate it. For both the plots, the paramters are: $\epsilon_i = 10^3$, $\Delta\eta_i = 190$ and $J_i = 16$.

4.1.1 The Attack Model

An offline setting oversees the threat model of this attack. The adversary has access to the outputs of the Laplace mechanism used in the ϵ-Scheduler. More application-specific details are in Section 4.2. We then show that the adversary can use the attack with the observed output to get a good prediction of the noise being used for each job in a task. Since the adversary gets to see the schedule and the "secure" output, it will know the arrival times of all the jobs that it has already seen. However, crucially, it has no information on the arrival times of future jobs and this is what it is trying to guess. We also assume that the adversary knows the mechanism being employed to achieve schedule indistinguishability, *i.e.*, the Laplace mechanism. In summary, the adversary:

- knows the output after randomization
- can measure the time it takes to compute the output and knows the implicit periods

Any attack requires the adversary to know a baseline of running times and their corresponding noise magnitudes; so it is able to time many executions of schedule indistinguishability acting on a task. In the differential privacy case, this is akin to the adversary measuring the time taken to return a query, where the query returns the noise induced output. Using the baseline values, the adversary has to make a prediction on an unknown noise. Thus, it needs just one observation after its baseline calculations.

This attack has some limitations. First, we assume that the time taken to produce the output by the ϵ-Scheduler directly represents the time taken to do the sampling. There may be other moving parts that add to the time of the function that returns the noise induced

value. Second, this attack will largely fail if the attacker cannot measure the time taken to generate a single noise value. In implementations where this is not possible, the time taken to generate multiple noise values would not be as useful.

4.2 ANALYSIS

In our experiments the baseline values are obtained by aggregating around 10,000 values for each discrete noise magnitude in the range of interest. This is done separately for the randomized look-up method and the geometric sampling method. To get the precise times, we can use the `clocks_per_sec()` method with `clock()` or simply the `time()` method in the `get_laplace_inter_arrival_time()` function [33] with averaging to maintain a precision in microseconds. In video streaming, an adversary may also analyze the traffic capture file to determine when every packet has arrived. The Laplace distribution is symmetric and hence the time taken to draw negative and positive values of the same magnitude is also the same. Figure 4.1 shows the average times required to draw integer noise values with both methods of noise generation, where the noise magnitude range was $[0, 50]$. It is clear that the magnitude of the noise has a positive, mostly linear relationship with the time taken for the noise induced value to return. We assume that the adversary does not know the range of integer values of noise, like $[0, 9]$ for the experiments below (the adversary calculates the baseline values for as many magnitudes as it can). We ran 1,000 trials for each sampling technique to find the success of the attack. The attacker is trying to find x when $y = x + j$, where x is the arrival time, y is the randomized arrival time and j is the noise. The attack has the following steps:

1. Calculate the average times t_i for generating noise magnitudes $i \in [0, 9]$ (in $100\mu s$), where the sign of the noise being added is randomly determined.
2. Measure time t_j taken to generate the unknown noise j
3. Choose i that has closest time to t_j as the guess, *i.e.*, $j_{guess} = \arg\min_{i \in [0,9]}(|t_j - t_i|)$

There are two possible matching techniques: exact match, where $|j| = j_{guess}$ and approximate match, where $-1 \leq j_{guess} - |j| \leq 1$. So the possible values for exact match are $\pm j_{guess}$, and the possible values for approximate match are $\pm j_{guess}$, $\pm(j_{guess} - 1)$ and $\pm(j_{guess} + 1)$. Remember that the sign of the noise cannot be guessed by the adversary, so uncertainty still remains. Let the upper and lower bounds on the Laplace noise be u and l, respectively. Then, the range of the guess space for the attacker goes from $\frac{1}{u-l-1}$ to $\frac{1}{2}$ when exact matching is used. The guess space goes from $\frac{1}{u-l-1}$ to $\frac{1}{6}$ when approximate matching

Table 4.1: Success rates of timing side-channel attacks against implementations of discrete Laplace distributions. The baseline accuracy based on a random guess for exact and approximate match is 10% and 33%, respectively. We observe that the attack accuracy is always above the corresponding baseline accuracy.

ϵ	Sampling	Match	Accuracy
1000	lookup	exact	13.2%
		approximate	37.1%
	geometric	exact	25.3%
		approximate	45.0%
100	lookup	exact	12.5%
		approximate	33.1%
	geometric	exact	18.6%
		approximate	39.7%

is used. For example, consider that the attacker wants to know the true arrival time that is $x = 10$, but the noise induced arrival time is $y = 15$. After observing the time taken to return $y = 15$, the adversary guesses -6. This is a success under approximate matching as $j = -(j_{guess} + 1) = 5$, but it's not a success under exact matching. The results are shown in Table 4.1, where $\epsilon \in \{100, 1000\}$, $\Delta\eta_i = 190$ and $J_i = 16$. Since the attacker tries both the positive and negative values, the baseline accuracy is 10 % for exact matching and 33 % for approximate matching. As seen in the table, the success of the attack is always above the baseline values. Thus, we conclude that observing the time taken to return the noise induced output leaks critical information like the relative magnitude of the noise used.

Comparatively, the look-up method is more immune to this attack as its time is mostly made up of the random number generator existing in the Linux kernel. Here, instead of directly sampling for the magnitude of the noise, sampling is done to find the index of the array that consists of the noise values. In the geometric method, since the time is directly calculated for the sampling method, the correlation is stronger and hence the accuracy of the attack is greater. We can gather that with larger noise values, the success of the attack decreases. This also means that the success of the attack will reduce as the Laplace distribution scale, b, keeps increasing.

4.3 RELATION TO SCHEDULE INDISTINGUISHABILTY

Next, we tried to gauge to what degree the knowledge of the noise actually helps an adversary in attacking the larger real-time system. For this purpose, we first measured the success of the video identification attack detailed in Chapter 3, when combined with this attack. Then, in order to make this analysis application-agnostic, we tried to come up with

Table 4.2: Success rates of the video identification attack after the traffic pattern is re-adjusted with noise obtained from the timing-based attack. The identification is correct 5 out of 15 times, which is a lot better than the performance in Chapter 3. These results are for the geometric sampling done in ϵ-Scheduler.

Test	lowest score	score with vid 1	result
1	0.0232	0.0256	fail
2	0.0217	0.0283	fail
3	0.0251	0.0302	fail
4	0.0242	0.0242	pass
5	0.0256	0.0269	fail
6	0.0236	0.0236	pass
7	0.0263	0.0271	fail
8	0.0221	0.0314	fail
9	0.0239	0.0239	pass
10	0.0218	0.0218	pass
11	0.0234	0.0287	fail
12	0.0201	0.0238	fail
13	0.0198	0.0253	fail
14	0.0227	0.0227	pass
15	0.0219	0.0264	fail

a generic framework that will tell us how much the adversary has benefited from knowing the noise values. For these experiments, the noise prediction attack was carried out for up to 300 magnitudes of integer noise (1 magnitude = 100 μs). When measuring the success of the video identification attack in the video streaming application, the traffic pattern was re-adjusted after the noise predicted by the attack in this section was removed. We did this by randomly adding or subtracting the predicted noise to or from the output arrival time of every frame, since our attack only gives the magnitude. The removal for every frame was done by analyzing the time information rendered by the `pcapng` file produced by Wireshark.

It is important to remember that the traffic pattern is in bits per second (bps), so we had to combine the new arrival times (after removing the noise) with the resolution of every frame, to calculate the new number of bits that were transmitted every second. This new traffic pattern was used to conduct the video identification attack. We kept the parameters of the video identification attack the same as in Chapter 3. The experiments were repeated 15 times when video 1 was being streamed. We predicted the noise for every frame that arrived within the eavesdropping period. Table 4.2 shows that the correct video was identified in 5 out of 15 tests when geometric sampling was done in ϵ-Scheduler. We observed 4 out of 15 successes when the lookup style sampling was done in ϵ-Scheduler. These are significant jumps from the 2 out of 15 successes shown in Figure 3.4.

Table 4.3: Success rates of determining the correct time slots after the noise is removed using values ascertained from the timing side-channel attack with look-up sampling.

$l_{slot}(ms)$	no. of slots	success rate
10	3000	$396/777 = 0.5096$
5	6000	$153/777 = 0.1969$
1	30000	$42/777 = 0.05405$

For our second experiment, we broke the time range of interest into n slots, each of length l_{slot}. Here, the goal of the adversary is *to predict in which time slot each job will arrive in the presence of the ϵ-Scheduler*. This goal is independent of the application concerned and hence tells more about how knowing the noise used for randomization in schedule indistinguishability poses a danger for RTS. The prediction is done after noise is calculated for every job based on the timer. The noise is then randomly added to or subtracted from a vanilla model with a constant period known to the adversary. To gauge the degree of success, the predicted slots are matched against the ground truth values (actual time slots in which the jobs arrived with ϵ-Scheduler) that were obtained from the video streaming application. The accuracy of the experiment is defined as the number of jobs for which slots were correctly predicted divided by the total number of jobs. We conducted our experiments with just $\epsilon = 1000$ due to time constraints. It would be interesting to see how the accuracy changes when we vary ϵ. We used 30 seconds as the observation time and $33.33ms$ as the period; all other values were the same as in Chapter 3.

From the results in Table 4.3, it is immediately clear that the accuracy heavily depends on the length of the time slot. We can gauge from the table that a large l_{slot} results in increased accuracy of the attack. Due to the modeling of the problem, it is easy to see that the experiment is far more likely to return correct predictions for the first few jobs rather than the latter jobs. This is because correct prediction of the latter jobs depend, to a degree, on the correct prediction of the former jobs as the time slots are ordered by their starting times. It is also hard to establish a baseline here due to the dependent nature of correct predictions for jobs. A small l_{slot} along with a high accuracy value means that the noise learned from the sampling based attack is useful in breaking schedule indistinguishability, thereby compromising the security of RTS.

4.4 POSSIBLE REMEDIES

It would be possible to stop the timing attack by either preventing the adversary from observing the time taken to produce a randomized arrival time or in the case where this is

Table 4.4: Success rates of timing side-channel attacks using a time delay for every draw. We observe that the attack accuracy has drastically reduced.

ϵ	Sampling	Match	Accuracy
1000	lookup	exact	8.5%
		approximate	29.3%
	geometric	exact	9.2%
		approximate	31.4%

impossible, by disrupting the time taken to return the output. The first method is out of scope mainly because it is an application-specific consideration. The second method can be done in two ways:

1. Repeat the sampling for a minimum number of times so that the noise magnitude is not related to the time taken. If we get a correct sample before this minimum, draw "blank" samples to reach the minimum.

2. Add a time delay for each draw to reach a sufficiently large time threshold, so that the time taken is constant for all discrete integer noise.

The above method will work for both the ways we have sampled the Laplace distribution in the ϵ-Scheduler. We tried the second method to test the theory. We fixed the time threshold to be $200\mu s$ because this value was larger than 99% of the times taken to return outputs. The accuracy results in Table 4.4 show a big drop in success of the attack, which is conceivable since we are essentially eliminating the side-channel using this approach. The obvious caveat of this method is that regardless of the noise being added, every draw will take at least $200\mu s$, thereby increasing the overall time of the randomization process which directly deteriorates the performance of the RTS. However, with a properly tuned time threshold that has an acceptable drop in performance, we can greatly reduce the success of the time-based attack.

CHAPTER 5: A PRIVACY BUDGET ATTACK

In this attack, we will demonstrate the presence of a side-channel that compromises schedule indistinguishability. In differential privacy terms, the side-channel can be exploited by an adversarial querier to mount an attack that weakens privacy [48]. Hence, this side-channel is of interest in schedule indistinguishability as it draws its main principles from differential privacy. The side-channel arises from the notion of a "privacy budget" that is present in differential privacy and thus, in schedule indistinguishability. As mentioned in Chapter 3, the scale of the Laplace noise introduced depends on the number of jobs we want to protect in our real-time system. This number, J, is a privacy budget that is ingrained within the ϵ-Scheduler. J can be further converted into the protection duration, as shown by Equation 3.6. Privacy budget can be thought of as a limit on the amount of information that is available to an adversary. Like in the Chapter 4, this attack model is also kept general enough so that it can be applied to any application using RTS. Again, we check the validity of this attack on the video streaming application and go on to suggest possible remedies.

5.1 DEFINITION AND SCOPE

Remember that the adversary can break schedule indistinguishability if it gets to observe the schedule for a very long time. This is because schedule indistinguishability degrades with increasing number of jobs and therefore with increasing observation time, making it easier for the adversary to predict future arrival times of jobs. This would translate to an aggregation of k queries violating ϵ-differential privacy as $\sum_i^k \epsilon_i > \epsilon$, where ϵ_i is the indistinguishability of the ith query. Hence, a real-time system with schedule indistinguishability should strive to keep a privacy timer that allows the schedule to be observed as long as the observation time does not greatly exceed the protection duration. The protection duration, that is provided by ϵ-indistinguishability [6], is a limit on the number of jobs for which job-level ϵ-indistinguishability holds. Alternatively, there can also be a privacy cost counter that keeps track of the number of jobs that have arrived and aborts the execution of the application when the number of jobs that have arrived is a lot more than the schedule indistinguishability limit on the number of jobs.

In the video streaming application, the protection duration is analogous to the eavesdropping window of time where the adversary gets to observe the network traffic. The protection duration can be thought of as something like the time between two security checks in a real-time system. Therefore, we need to ensure that the system is safe between these two

Table 5.1: There were a total of 20 tests with $\epsilon = 1000$ and protection duration (λ) = 500 ms, where 10 tests were with eavesdropping time = 180 seconds, and 10 tests with eavesdropping time = 5 seconds. We observe that eavesdropping for much longer than the protection duration undermines schedule indistinguishability.

eavesdropping window	protection duration (λ)	number of tests	number of passes
$180s$	$500ms$	10	9
$5s$	$500ms$	10	2

checks. Schedule indistinguishability relies on the assumption that an adversary can only observe the schedule and nothing else. But, as we will see, it can also observe side-channels, that leak other parameters tied to the schedule, like the time taken to return an output, detailed in Chapter 4, and the protection duration, detailed in this chapter.

The adversary's observation time should not exceed the protection duration. In most cases, however, if it exceeds by a small amount, then the system may still be protected from scheduler side-channels for practical purposes solely due to the randomization that disrupts the periodic nature of execution. But, if it is a lot greater than the protection duration, then the adversary will almost certainly be able to predict the arrival times of future jobs. We can keep a check on this by stopping the execution of the application as soon as the observation time goes past a predetermined application-specific threshold. In terms of differential privacy, this is similar to calculating the privacy cost of each query and then stopping any new queries from running if a budget threshold is surpassed. Here, it is important to point out that this threshold, in no way, is related to the actual arrival times of jobs. Thus, the threshold does not leak information about the arrival times themselves, but can leak information about the scale of the noise distribution being used, as we will see in Section 5.2.

We conducted some experiments on the video streaming application to show the importance of the protection duration. The video being run during the eavesdropping period for all tests was video 1. We ran the video identification attack from Chapter 3 on the captured traffic. From Table 5.1, we see that in the first set of 10 tests the correct video was identified 9 times out of 10 because the eavesdropping time ($180s$) greatly exceeded the protection duration ($500ms$), while in the second set the correct video was identified only twice out of 10 times. Evidently, the eavesdropping time cannot overshoot the protection duration provided by schedule indistinguishability by a lot. Surpassing it by a relatively small amount does not harm the security of the system here ($5s > 500ms$); but we may just be getting fortunate in these experiments because 5 seconds is too less an eavesdropping period for the matching technique to work in the video identification attack.

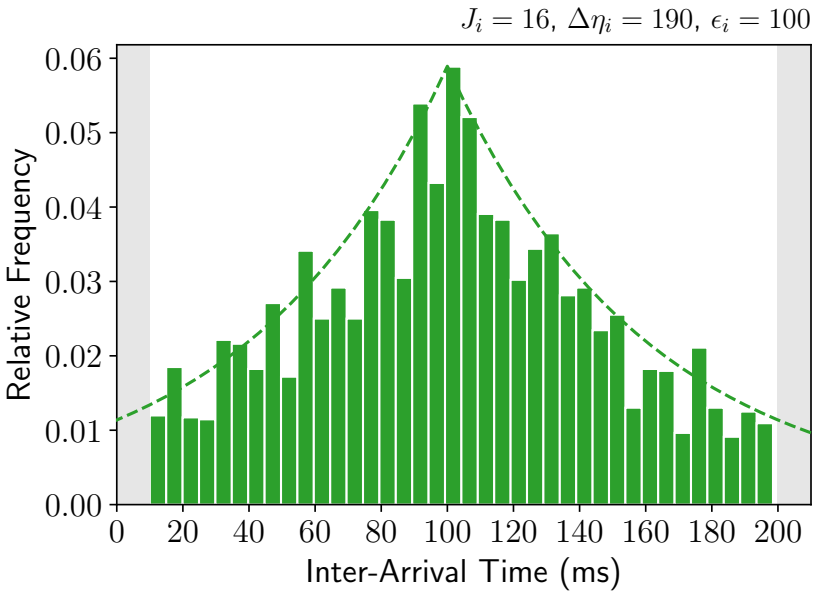


Figure 5.1: Reused from Chen *et al.* [6]. Histogram of the randomized inter-arrival times generated by ϵ-Scheduler for the task τ_i with a desired period $100ms$ running in RT Linux. The extended task parameters are assigned to be $\epsilon_i = 100$, $\Delta\eta_i = 190$ and $J_i = 16$ (the same as that shown in Figure 3.2(b)). The plot shows that the generated inter-arrival times are distributed under the desired Laplace distribution indicated by the dashed line.

5.1.1 The Attack Model

Consider that the system has a privacy budget counter that aborts the application if the budget is exceeded by some degree. In this case, the adversary has two pieces of information: the schedule for the observation period and the time after the start of observation at which the application stops. The fact that the application stopped executing at a certain point can be used to leak private information because essentially, this time is a result that the system gives back to the attacker. When this information is combined with the knowledge of the inter-arrival time function (easily obtainable from the system dynamics) and the bounds on the Laplace noise (available to the attacker), the attacker can determine J using Equation 3.6. All that is required to ascertain is ϵ, where the jobs are ϵ-indistinguishable.

For this the attacker runs a search with the most commonly used ϵ values in schedule indistinguishability for RTS. The search technique used is based on dynamic time warping (DTW) [52], similar to the matching technique used for the video identification attack in Chapter 3, but here we don't have to use p-DTW [46] because the arrival times of jobs do not depend on the work they do or their content. So long as the two times used for matching are the same, the specific job numbers that arrive in the observed schedule are irrelevant. This method will work because the probability density of the observed randomized inter-

arrival times closely resembles the Laplace distribution used for randomization as shown in Figures 5.1 and 3.2(b). Another alternative search method could be to directly fit the observed probability distribution with a Laplace distribution and check the goodness of the fit using the Kolmogorov-Smirnov test [53].

This attack has some limitations. First, we assume that the time when the application stops is close to the actual protection duration of the system. Without this assumption, the attack is not feasible. An incorrect value of J obtained from a λ that is much larger than the actual λ (or protection duration) will not produce good results when doing the matching-based search for determining the value of ϵ. Further, we assume that the attacker can accurately determine η from the set of admissible periods $\mathcal{T}$. Only then can the attacker accurately calculate $\Delta\eta$ that is required to get the scale of the Laplace distribution. The accuracy of any matching technique will be low if the above two parameters are not precisely determined. In implementations where this is not possible, the DTW matching will return ϵ values that are not useful for conducting other attacks.

5.2 ANALYSIS

We implement the attack in a way such that the privacy budget is revealed to the attacker, given the observation time is longer than the budget threshold. The attacker has to merely time its observation session. This allows the adversary to convert the budget in milliseconds to the number of job instances for which protection is guaranteed by schedule indistinguishability. Thus, with the value of J determined, the adversary proceeds to find $\Delta\eta$. The adversary has been observing the schedule for long enough for the privacy budget to expire. It is also assumed that the adversary is given access to the set of admissible periods for the task, via the system dynamics. The adversary can learn the inter-arrival time function from this. This may not always be straightforward, but we assume this holds true for our experiments. Once the inter-arrival time function is determined, calculating the inter-arrival time sensitivity, $\Delta\eta$, is not difficult. Armed with the values of J and $\Delta\eta$, the adversary starts the search for the correct ϵ. It uses a temporal matching technique called DTW [52] to obtain its guess of the correct ϵ. The matching is done between the observed points of job arrivals (obtained via the observation period) and the sample arrival time model for the same duration using a scale determined by the trial ϵ value and the known values of J and $\Delta\eta$. The ϵ values used in the search are $1, 10, 10^2$ and 10^3 because these are the reasonable ϵ values for RTS as suggested by Figure 3.2(a).

For the above to work, the attacker has to focus only on the arrival times it has seen during the observation period. With a large enough observation time, the distribution

Table 5.2: Success rates of privacy budget attacks against different values of ϵ. The baseline accuracy based on a random guess for the ϵ is 25% because we use four candidate values for ϵ: $\{1, 10, 10^2, 10^3\}$. We observe that the attack is successful in every test.

true ϵ	d^*	$d_{\epsilon=1}$	$d_{\epsilon=10}$	$d_{\epsilon=10^2}$	$d_{\epsilon=10^3}$
10^3	0.36	0.07	0.09	0.13	0.36
	0.47	0.05	0.06	0.10	0.47
	0.42	0.09	0.09	0.19	0.42
10^2	0.33	0.12	0.21	0.33	0.16
	0.29	0.11	0.15	0.29	0.19
	0.31	0.04	0.12	0.31	0.13
10	0.43	0.22	0.43	0.25	0.11
	0.44	0.18	0.44	0.12	0.11
	0.39	0.17	0.39	0.16	0.08
1	0.48	0.48	0.31	0.09	0.06
	0.52	0.52	0.3	0.07	0.07
	0.43	0.43	0.32	0.15	0.08

exhibited by the observed randomized inter-arrival timing pattern will be very similar to the Laplace distribution, with scale $b = \frac{2J\Delta\eta}{\epsilon}$, used for randomization. Therefore, the attacker simply measures the similarity between the two temporal sequences: the sequences of the randomized inter-arrival times from the observed schedule and the sequences of a sample Laplace distribution with $\epsilon \in \{1, 10, 10^2, 10^3\}$, when J and $\Delta\eta$ are known. We will get four similarity scores, $d_{\epsilon=1}$, $d_{\epsilon=10}$, $d_{\epsilon=10^2}$ and $d_{\epsilon=10^3}$, since we are trialing four possible values of ϵ. The method returns the ϵ with the closest match, *i.e.*, the ϵ corresponding to the lowest normalized similarity score, $d^* = min\{d_{\epsilon=1}, d_{\epsilon=10}, d_{\epsilon=10^2}, d_{\epsilon=10^3}\}$. The similarity score is an indicator of the ϵ that is likely to produce the observed timing pattern, when combined with the already determined values of J and $\Delta\eta$. A success in the above implementation of the privacy budget attack is when $d^* = d_{\epsilon=y}$, where y is the real value of ϵ in ϵ-indistinguishability, *i.e.*, in the ϵ-Scheduler, $\epsilon = y$. The noise generation technique does not matter for this attack, and hence, for all the experiments the default look-up method of generating noise in ϵ-Scheduler was used.

We conducted this experiment across 3 tests for each ϵ value where the budget threshold is equal to λ and a different schedule is considered for every test. We can qualitatively see the similarity between the observed probability density and the correct Laplace probability density in Figure 5.1, where the raw arrival times are used to generate the observed probability density. We obtain similarity scores via DTW by simply using the two temporal sequences: one observed from the schedule and one sampled from the candidate Laplace distribution. The results are shown in Table 5.2, where $\epsilon \in \{1, 10, 10^2, 10^3\}$, $\Delta\eta = 190$ and $J = 16$.

The baseline accuracy for this attack is 25% as there are four possible values the adversary

tests for. As seen in the table, *we could successfully determine the correct ϵ in all the 12 tests.* Thus, we learn that observing time required to exhaust the privacy budget of a task with schedule indistinguishability can potentially leak vital information like the ϵ parameter of the Laplace distribution used to add noise to the job instances of the task. One would assume that since the relative probability distributions of the candidate ϵ values are so different that DTW is bound to work well with the observed schedule. However, probability distributions for ϵ values that are close to one another are similar and hence, the success of this attack will surely reduce if the candidate ϵ's are very close to one another. Moreover, in that case, predicting an ϵ value close to the correct ϵ value may be enough to compromise schedule indistinguishability and hence it is not considered in this thesis.

5.3 RELATION TO SCHEDULE INDISTINGUISHABILTY

Similar to Chapter 4, we tried to gauge how much knowledge of ϵ in ϵ-indistinguishability actually helps an adversary in attacking the larger real-time system. For this purpose we first measured the success of the identification attack detailed in Chapter 3, when combined with this attack. Then, in order to make this analysis application-agnostic, we tried to come up with a generic framework that will tell us how much the adversary has benefited from knowing the ϵ values. For these experiments, the privacy budget attack was carried out for the four candidate ϵ values mentioned in the previous section. When measuring the success of the video identification attack in the video streaming application, the traffic pattern was re-adjusted. This is done by removing a sample noise sequence of the same length (in terms of the number of frames that arrived during the observation) that is derived from the Laplace distribution with the ϵ predicted by the attack, with $\mu = 0$. In contrast to the timing-based attack, we do not know the exact noise values, hence a sample noise sequence is the best we can do. This makes the attack less suited to work with the video identification.

We removed the noise by subtracting the corresponding noise values in the noise sequence from the output arrival times of every frame. The noise removal for every frame was done by analyzing the time information rendered by the `pcapng` file captured using Wireshark. It is important to remember that the traffic pattern is in bps, so we had to combine the new arrival times (after removing the noise) with the resolution of every frame to calculate the new number of bits that were transmitted every second. This new traffic pattern was used as input to the video identification attack. We kept the parameters of the video identification the same as in Chapter 3. The experiments were repeated 15 times when video 1 was being streamed. We used a value from the noise sequence for every frame that arrived within the eavesdropping period. Table 5.3 shows that the correct video was identified in 2 out of the

Table 5.3: Success rates of the video identification attack after the traffic is re-adjusted with noise taken from a noise sequence that is generated using the predicted ϵ value from the privacy budget attack. The identification is correct 2 out of 15 times, same as the performance in Chapter 3.

Test	lowest score	score with vid 1	result
1	0.0212	0.0293	fail
2	0.0218	0.0241	fail
3	0.0234	0.0285	fail
4	0.0246	0.0306	fail
5	0.0225	0.0263	fail
6	0.0241	0.0241	pass
7	0.0267	0.0296	fail
8	0.0212	0.0224	fail
9	0.0233	0.0309	fail
10	0.0239	0.0270	fail
11	0.0222	0.0253	fail
12	0.0234	0.0234	pass
13	0.0249	0.0254	fail
14	0.0223	0.0298	fail
15	0.0247	0.0286	fail

15 tests. This ratio is same as the 2 out of 15 result shown in Figure 3.4. Therefore, this attack, while dangerous for schedule indistinguishability, does not work quite well with the video identification attack, when the above implementation is carried out.

For our second experiment, we broke the time range of interest into n slots, each of length l_{slot}, like in Chapter 4. The goal of the adversary is *to predict in which time slot each job will arrive in the presence of the ϵ-Scheduler*. This goal is independent of the application concerned and hence tells more about how knowing the ϵ used in schedule indistinguishability poses a danger to RTS. The prediction is done after noise is calculated from a noise sequence derived from the Laplace distribution with the predicted ϵ value. This is done for every job in the time range. The noise is then subtracted from a vanilla model with a constant period known to the adversary. To gauge the degree of success, the predicted slots are matched against the ground truth values (actual time slots in which the jobs arrived with ϵ-Scheduler), that we obtained from the video streaming application. The accuracy of the experiment is defined as the number of jobs for which slots were correctly predicted divided by the total number of jobs. Like in the previous chapter, we used 30 seconds as the observation time and 33.33 ms as the period; all other values were the same as in Chapter 3. We conducted our experiments with just $\epsilon = 1000$ due to time constraints. It would be interesting to see how the accuracy changes when we vary ϵ.

Table 5.4: Success rates of determining the correct time slots after the noise is removed using values taken from a noise sequence that is generated using the predicted ϵ value from the privacy budget attack.

$l_{slot}(ms)$	no. of slots	success rate
10	3000	$184/777 = 0.2368$
5	6000	$57/777 = 0.07336$
1	30000	$15/777 = 0.01930$

Again we see from Table 5.4 that the accuracy heavily depends on the length of the time slot, and is less than the accuracy of the timing side-channel attack for the same l_{slot} value. Larger the length of the time slot, more the accuracy of the attack. Large time slots, however, are not useful to the adversary. More details on the characteristics of this attack are mentioned in Chapter 4, Section 3. We do not have a baseline here due to the dependence of correct predictions of future jobs on correct predictions of past jobs. Presumably, a small time slot length with a high accuracy value means that the ϵ learned from the privacy budget attack is useful in breaking schedule indistinguishability, thereby compromising the security of RTS. This attack performs worse than the timing side-channel attack in both the experiments mostly because randomly obtaining noise values from a sequence drawn from $\text{Lap}(\frac{2J\Delta\eta}{\epsilon})$ has a lot of uncertainty even if the scale of the Laplace distribution is accurate.

5.4 POSSIBLE REMEDIES

For systems that have a privacy budget, if we can figure out a way to stop the adversary from observing the time when the privacy budget is exhausted, then we can stop this attack. An easy fix would be to prevent the adversary from observing the schedule for that long, where the system stops execution when the adversary has crossed an observation time threshold that is less than and far from the protection duration. However, this is not feasible in all real-time applications. For instance, in the video streaming application with $\lambda = 500ms$, the application has to restart after every time t (where $t < 500ms$) to prevent the adversary from eavesdropping for as long as the protection duration, λ. Such a design decision is not practical for a video streaming application where videos are several minutes long. Conversely, we cannot increase λ to be in minutes as that will greatly increase the scale of the Laplace distribution and thus, greatly decrease the real-time utility of the application (measured in terms of frames per second).

When the adversary's observations cannot be controlled, it would seem that the only way to stop the adversary from succeeding is to reduce the effectiveness of the matching method. One way to do this is by changing the ϵ value for the Laplace noise every j number of job

instances. Now the adversary searching for the correct ϵ value using DTW, will not succeed as the scheduler uses more than one ϵ value to add noise. The more the number of ϵ values used and the more frequently they are alternated, the less will be the success of the privacy budget attack. Although, this method is promising, it is likely to incur more computational overhead and will surely deteriorate the utility of the real-time application. Parameters to be taken into consideration include: the number of ϵ values used for randomization and the number of jobs, j, after which we change the ϵ value. Both will be tied to the desired QoS of the application and hence achieving a trade-off is key. With properly tuned parameter values that cause an acceptable drop in performance, we can greatly reduce the success of a privacy budget attack.

CHAPTER 6: CONCLUSIONS

My primary goal in this thesis was to present schedule indistinguishability as a way to prevent scheduler side-channels and show some vulnerabilities in its implementation via the ϵ-Scheduler. To this end, I started off by discussing the problems that RTS face, with particular emphasis on scheduler side-channels. Then, I presented an overview of schedule indistinguishability that acts as a countermeasure against scheduler side-channels and showed its effectiveness in preventing an identification attack on a real-world video streaming application. I gave a detailed summary of the matching method used to carry out the identification attack. An adversary who can observe the network traffic and has the video fingerprints can use the matching method to mount the identification attack.

In the remaining portion of the thesis, I explored the scope of attacks that depend on vulnerabilities in schedule indistinguishability due to side-channels. While side-channels do not reveal the private timing information of the schedule, *i.e.*, the arrival times of jobs, they leak other information about the private data. Attacks that exploit these side-channels can compromise schedule indistinguishability, thereby weakening security against scheduler side-channels in RTS. I presented two such attacks on the ϵ-Scheduler. The first attack is based on a timing side-channel that leaks the amount of time taken by the sampler to produce a noise sample from the Laplace distribution. I showed that the time taken has a positive linear relationship to the magnitude of noise sampled using two methods of sampling: the default look-up method and a geometric sampling method. The second attack uses the value of the protection duration, guaranteed by schedule indistinguishability, to predict the ϵ parameter. I showed that by carrying out a simple search, an adversary can estimate the ϵ value with high certainty.

I evaluated the utility of each attack in negating the protection provided by schedule indistinguishability using the identification attack on the video streaming application. Furthermore, in order to make the evaluation generic to any real-time application, I conducted tests on a time slot based evaluation framework. The results reveal that side-channel attacks have the potential to undermine the security provided by schedule indistinguishability in RTS. For both the attacks, I suggest some possible remedies that can make the ϵ-Scheduler immune to side-channels. Finally, with the research and experiment results presented in this thesis, the hypothesis - *there exist vulnerabilities in schedule indistinguishability applied to RTS due to side-channels leaking critical information about the private data*, can be validated.

6.1 FUTURE WORK

The most immediate future work involves running the side-channel attacks in an online setting. While the offline setting, as presented in this thesis, provides good tools for analysis, it is far from realistic. In addition to evaluation in an online setting, it is important to conduct experiments using other real-world applications like the rover application in Chen *et al.* [6], to gauge the viability of the attacks. Even if we block the two side-channels presented, the noise-induced arrival times in the schedule can themselves be a source of information leakage. After observing a sufficient number of arrival times, the adversary can utilize a distribution learning framework [54] to predict the distribution used and its scale.

Since the side-channels reveal some disadvantages of the ϵ-Scheduler, it is possible that noise-based randomization of schedules is not the best way to realize schedule indistinguishability. One could design the scenario as a cryptography-style game between an adversary and a challenger, where both have clearly defined goals and advantages. The game proceeds in the style of a membership inference attack [55] or a deletion inference attack [56]. Hence, there may be an opportunity to employ the principles used to safeguard against membership/deletion inference attacks, in our domain. In the randomization technique itself, we can postulate other ways of adding noise, *e.g.*, adding it to schedule components other than the arrival times, like the deadlines and execution times.

REFERENCES

[1] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham et al., "Experimental security analysis of a modern automobile," in *2010 IEEE symposium on security and privacy*. IEEE, 2010, pp. 447–462.

[2] D. U. Case, "Analysis of the cyber attack on the ukrainian power grid," *Electricity Information Sharing and Analysis Center (E-ISAC)*, vol. 388, pp. 1–23, 2016.

[3] T. M. Chen and S. Abu-Nimeh, "Lessons from stuxnet," *Computer*, vol. 44, no. 4, pp. 91–93, 2011.

[4] B. Min and V. Varadharajan, "Design and analysis of security attacks against critical smart grid infrastructures," in *2014 19th International Conference on Engineering of Complex Computer Systems*. IEEE, 2014, pp. 59–68.

[5] M.-K. Yoon, B. Liu, N. Hovakimyan, and L. Sha, "Virtualdrone: virtual sensing, actuation, and communication for attack-resilient unmanned aerial systems," in *Proceedings of the 8th international conference on cyber-physical systems*, 2017, pp. 143–154.

[6] C.-Y. Chen, D. Sanyal, and S. Mohan, "Indistinguishability prevents scheduler side channels in real-time systems," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 666–684.

[7] K. Jiang, L. Batina, P. Eles, and Z. Peng, "Robustness analysis of real-time scheduling against differential power analysis attacks," in *2014 IEEE Computer Society Annual Symposium on VLSI*. IEEE, 2014, pp. 450–455.

[8] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan, "The sorcerer's apprentice guide to fault attacks," *Proceedings of the IEEE*, vol. 94, no. 2, pp. 370–382, 2006.

[9] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems," in *Annual International Cryptology Conference*. Springer, 1996, pp. 104–113.

[10] C.-Y. Chen, S. Mohan, R. Pellizzoni, R. B. Bobba, and N. Kiyavash, "A novel side-channel in real-time schedulers," in *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2019, pp. 90–102.

[11] J. Son et al., "Covert timing channel analysis of rate monotonic real-time scheduling algorithm in mls systems," in *2006 IEEE Information Assurance Workshop*. IEEE, 2006, pp. 361–368.

[12] H. Baek and C. M. Kang, "Scheduling randomization protocol to improve schedule entropy for multiprocessor real-time systems," *Symmetry*, vol. 12, no. 5, p. 753, 2020.

[13] K. Krüger, M. Volp, and G. Fohler, "Vulnerability analysis and mitigation of directed timing inference based attacks on time-triggered systems," *LIPIcs-Leibniz International Proceedings in Informatics*, vol. 106, p. 22, 2018.

[14] M.-K. Yoon, S. Mohan, C.-Y. Chen, and L. Sha, "Taskshuffler: A schedule randomization protocol for obfuscation against timing inference attacks in real-time systems," in *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2016, pp. 1–12.

[15] C. Dwork, "Differential privacy: A survey of results," in *International conference on theory and applications of models of computation*. Springer, 2008, pp. 1–19.

[16] C. Dwork, A. Roth et al., "The algorithmic foundations of differential privacy," *Foundations and Trends® in Theoretical Computer Science*, vol. 9, no. 3–4, pp. 211–407, 2014.

[17] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM (JACM)*, vol. 20, no. 1, pp. 46–61, 1973.

[18] S. Kadloor, N. Kiyavash, and P. Venkitasubramaniam, "Mitigating timing based information leakage in shared schedulers," in *2012 Proceedings IEEE INFOCOM*. IEEE, 2012, pp. 1044–1052.

[19] X. Gong, N. Kiyavash, and P. Venkitasubramaniam, "Information theoretic analysis of side channel information leakage in fcfs schedulers," in *2011 IEEE International Symposium on Information Theory Proceedings*. IEEE, 2011, pp. 1255–1259.

[20] A. Easwaran, A. Chattopadhyay, and S. Bhasin, "A systematic security analysis of real-time cyber-physical systems," in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2017, pp. 206–213.

[21] C.-Y. Chen, M. Hasan, and S. Mohan, "Securing real-time internet-of-things," *Sensors*, vol. 18, no. 12, p. 4356, 2018.

[22] M.-K. Yoon, S. Mohan, J. Choi, and L. Sha, "Memory heat map: Anomaly detection in real-time embedded systems using memory behavior," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 2015, pp. 1–6.

[23] T. Xie and X. Qin, "Improving security for periodic tasks in embedded systems through scheduling," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 6, no. 3, pp. 20–es, 2007.

[24] M. Lin, L. Xu, L. T. Yang, X. Qin, N. Zheng, Z. Wu, and M. Qiu, "Static security optimization for real-time systems," *IEEE Transactions on Industrial Informatics*, vol. 5, no. 1, pp. 22–37, 2009.

[25] D. Trilla, C. Hernandez, J. Abella, and F. J. Cazorla, "Cache side-channel attacks and time-predictability in high-performance critical real-time systems," in *Proceedings of the 55th Annual Design Automation Conference*, 2018, pp. 1–6.

[26] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi, "The em side—channel (s)," in *International workshop on cryptographic hardware and embedded systems*. Springer, 2002, pp. 29–45.

[27] M. Völp, C.-J. Hamann, and H. Härtig, "Avoiding timing channels in fixed-priority schedulers," in *Proceedings of the 2008 ACM symposium on Information, computer and communications security*, 2008, pp. 44–55.

[28] N. Tsalis, E. Vasilellis, D. Mentzelioti, and T. Apostolopoulos, "A taxonomy of side channel attacks on critical infrastructures and relevant systems," in *Critical Infrastructure Security and Resilience*. Springer, 2019, pp. 283–313.

[29] M. Völp, B. Engel, C.-J. Hamann, and H. Härtig, "On confidentiality-preserving real-time locking protocols," in *2013 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2013, pp. 153–162.

[30] A. Ghassami, X. Gong, and N. Kiyavash, "Capacity limit of queueing timing channel in shared fcfs schedulers," in *2015 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2015, pp. 789–793.

[31] M. Nasri, T. Chantem, G. Bloom, and R. M. Gerdes, "On the pitfalls and vulnerabilities of schedule randomization against schedule-based attacks," in *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2019, pp. 103–116.

[32] K. Chatzikokolakis, M. E. Andrés, N. E. Bordenabe, and C. Palamidessi, "Broadening the scope of differential privacy using metrics," in *International Symposium on Privacy Enhancing Technologies Symposium*. Springer, 2013, pp. 82–102.

[33] C.-Y. Chen, "Epsilon-scheduler," https://github.com/synercys/Epsilon-Scheduler, accessed: 2022-04-14.

[34] M. Schwarz, M. Lipp, D. Gruss, S. Weiser, C. Maurice, R. Spreitzer, and S. Mangard, "Keydrown: Eliminating software-based keystroke timing side-channel attacks," in *Network and Distributed System Security Symposium*. Internet Society, 2018.

[35] J. He, L. Cai, and X. Guan, "Differential private noise adding mechanism and its application on consensus algorithm," *IEEE Transactions on Signal Processing*, vol. 68, pp. 4069–4082, 2020.

[36] F. Liu, "Generalized gaussian mechanism for differential privacy," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 4, pp. 747–756, 2018.

[37] H. S. Chwa, K. G. Shin, and J. Lee, "Closing the gap between stability and schedulability: A new task model for cyber-physical systems," in *2018 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, 2018, pp. 327–337.

[38] C. Lu, J. A. Stankovic, S. H. Son, and G. Tao, "Feedback control real-time scheduling: Framework, modeling, and algorithms," *Real-Time Systems*, vol. 23, no. 1, pp. 85–126, 2002.

[39] S. Liu, N. Guan, D. Ji, W. Liu, X. Liu, and W. Yi, "Leaking your engine speed by spectrum analysis of real-time scheduling sequences," *Journal of Systems Architecture*, vol. 97, pp. 455–466, 2019.

[40] F. D. McSherry, "Privacy integrated queries: an extensible platform for privacy-preserving data analysis," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, 2009, pp. 19–30.

[41] M. Hasan, S. Mohan, R. B. Bobba, and R. Pellizzoni, "Exploring opportunistic execution for integrating security into legacy hard real-time systems," in *2016 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, 2016, pp. 123–134.

[42] F. Abdi, C.-Y. Chen, M. Hasan, S. Liu, S. Mohan, and M. Caccamo, "Guaranteed physical security with restart-based design for cyber-physical systems," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, 2018, pp. 10–21.

[43] F. Abdi, C.-Y. Chen, M. Hasan, S. Liu, S. Mohan, and M. Caccamo, "Preserving physical safety under cyber attacks," *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 6285–6300, 2018.

[44] C. L. Canonne, G. Kamath, and T. Steinke, "The discrete gaussian for differential privacy," *Advances in Neural Information Processing Systems*, vol. 33, pp. 15 676–15 688, 2020.

[45] M. Grinberg, *Flask web development: developing web applications with python.* " O'Reilly Media, Inc.", 2018.

[46] J. Gu, J. Wang, Z. Yu, and K. Shen, "Traffic-based side-channel attack in video streaming," *IEEE/ACM Transactions on Networking*, vol. 27, no. 3, pp. 972–985, 2019.

[47] J. Jin, E. McMurtry, B. I. Rubinstein, and O. Ohrimenko, "Are we there yet? timing and floating-point attacks on differential privacy systems," *arXiv preprint arXiv:2112.05307*, 2021.

[48] A. Haeberlen, B. C. Pierce, and A. Narayan, "Differential privacy under fire," in *20th USENIX Security Symposium (USENIX Security 11)*, 2011.

[49] Google, "Secure noise generation," https://github.com/google/differential-privacy/blob/main/common_docs/Secure_Noise_Generation.pdf, accessed: 2022-04-14.

[50] I. Mironov, "On significance of the least significant bits for differential privacy," in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 650–661.

[51] IBM, "The discrete gaussian for differential privacy," https://github.com/IBM/discrete-gaussian-differential-privacy, accessed: 2022-04-14.

[52] D. J. Berndt and J. Clifford, "Using dynamic time warping to find patterns in time series." in *KDD workshop*, vol. 10, no. 16. Seattle, WA, USA:, 1994, pp. 359–370.

[53] F. J. Massey Jr, "The kolmogorov-smirnov test for goodness of fit," *Journal of the American statistical Association*, vol. 46, no. 253, pp. 68–78, 1951.

[54] I. Diakonikolas, "Learning structured distributions." *Handbook of Big Data*, vol. 267, pp. 10–1201, 2016.

[55] N. Carlini, S. Chien, M. Nasr, S. Song, A. Terzis, and F. Tramer, "Membership inference attacks from first principles," *arXiv preprint arXiv:2112.03570*, 2021.

[56] J. Gao, S. Garg, M. Mahmoody, and P. N. Vasudevan, "Deletion inference, reconstruction, and compliance in machine (un) learning," *arXiv preprint arXiv:2202.03460*, 2022.