Improving Security and Practicality of Modern Trusted Execution Environments

Background. Cloud computing underpins critical services in healthcare, finance, and AI. Yet, it faces a constant threat of cyberattacks and data breaches. In 2024, these cybercrimes exposed over 280 million records in the US's health sector alone. Data breaches are not only financially devastating for the organizations employing cloud services, costing them up to $4.4 billion in remediation on average in 2025, they also erode customers' trust in cloud providers.

To regain trust, cloud providers are now strengthening the security of their own infrastructure and isolating their customers' systems from it to minimize the risk of compromises in their platform from permeating to customers. In the latter case, particularly, providers are increasingly adopting specialized hardware from compute vendors (e.g., Intel, AMD). These are new processors with secure primitives built into the silicon that enable the creation of a secure "bubble" on the platform, called a Trusted Execution Environment (TEE), which protects sensitive data and programs, even if the rest of the system, including the cloud administrator, is compromised.

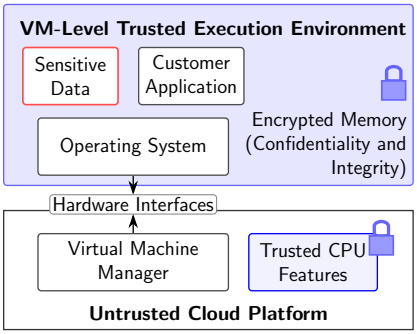

Figure 1: Modern TEE design

After the initial exploration of TEE designs, the community has settled on the idea of isolating an application along with its complete execution environment, including libraries and supporting software, within a single TEE. This design has helped migrate many applications into TEEs, resulting in reduced performance and engineering costs. Especially, companies have been deploying cutting-edge workloads such as large language model inference into TEEs with only trivial adaptation.

Challenges of modern TEEs. This direction in TEE design introduces two new security challenges. The first challenge lies in the huge amount of code now being placed inside a TEE, on which the security guarantees for the customer's application and data depend. This code (called the trusted computing base or the TCB of an application) includes the operating system, often consisting of several million lines of code and known to have thousands of vulnerabilities that the adversaries could exploit to compromise the TEE. Thus, making TEE code resilient against compromises is a key research challenge for the community.

Second, as a component of TEEs, the role of operating systems must now be rethought. Unlike normal applications, the operating system's primary task is to manage system resources, such as memory and files, by utilizing its access to low-level hardware mechanisms that provide fine control over software running on the system. Importantly, the operating system's high degree of control not only enables transparent resource management but also offers powerful tools for building in-TEE security mechanisms that are practical and performant. The work conducted during my doctoral studies demonstrated the promise of this approach, leveraging OS capabilities to implement practical TEE self-defense mechanisms, particularly against side-channel attacks. Building on that foundation, the challenge now is to explore a broader class of OS-supported techniques that address the issues in practical TEE deployments.

Toward taming TEEs' large TCB. Given the complexity of modern TEEs, securing the application TCB is becoming an essential challenge. Over the years, research has sought to tackle this challenge in three ways. The first approach proposes comprehensively testing the code placed inside the TEE to detect mistakes. Applying this to a large codebase, such as the OS, would require an insurmountable effort and is unlikely to find all the bugs. The second approach is compartmentalization, which splits a large codebase into smaller, isolated components. The intuition is that if each component is placed in a "cage" where failures cannot spread, then the system as a whole becomes more robust to attacks. Still, the decision of which code to place into components is largely unexplored for newer TEE designs. The third approach is TCB reduction, which seeks to minimize the code running inside the TEE, based on the belief that code size correlates with security risk. While this principle holds some truth, an exclusive focus on code size

is misleading. It can sacrifice system performance by removing safe features, and more importantly, it overlooks where the risky code actually resides.

I propose shifting the focus from the size of the TCB to the means by which trusted and untrusted worlds communicate – their interfaces. These interfaces, often inherited from legacy systems, were rarely designed with strong adversarial models in mind and are easy targets for attacks. Numerous studies have demonstrated that the interfaces expose critical vulnerabilities that leak sensitive data due to their lack of security considerations and legacy design choices. Building on this observation, I plan to pursue research in two directions.

First, building on my experience in software compartmentalization, I plan to explore methodologies that compartmentalize code interacting with the interfaces, thereby containing the impact of the untrusted platform on security where it matters most. A challenge would be determining the boundary of the compartment to maximize performance while minimizing possible attack surfaces. To this end, I plan to collaborate with researchers from Systopia Lab, who are currently developing theoretical models to evaluate compartmentalization strategies, thereby informing my proposed compartmentalized interface designs.

Second, I will revisit legacy TEE interface design choices, rethinking interfaces from the ground up with security as a first-class goal. This involves carefully analyzing the current interfaces and standards used by TEEs, evaluating the potential attack vectors, and proposing safer alternatives. This line of work will be based on a work-in-progress in collaboration with Hugo Lefeuvre and Margo Seltzer at UBC.

By focusing security decisions at the interfaces where they matter, TEE designs can manage complexity in a principled manner without sacrificing functionality. This enables the introduction of additional features, such as performance optimizations. More importantly, security mechanisms that are based on complex OS features can be implemented, as presented next.

Exploration of OS as a security component. My exploration in this direction will focus on practical challenges that plague modern TEE deployments: side-channel mitigation and private data processing in the cloud.

Practical side-channel mitigation Side channels are a critical yet persistent class of vulnerabilities at the software-hardware boundary of TEEs, allowing attackers to leak sensitive information from sensitive computations even without direct access. The specific class of side channels that are the focus of this proposal is those that leak sensitive applications' patterns of accessing resources (e.g., memory and CPU cache) that are shared between the attacker and the victim. For instance, while cloud platforms are prevented from accessing the TEE's memory, they can still move the encrypted memory to the disk (a process called swapping) to free up memory for other purposes. Control over the shared memory allows them to (1) read TEE's encrypted memory and (2) track the TEE's access to swapped-out memory. This introduces side channels referred to as the ciphertext side channels for the former, and controlled channels for the latter.

To illustrate the impact of these side channels, consider a TEE application that receives as input the user's age. The application itself handles data from users who belong to age groups by invoking different processing routines. While there is no direct exposure, the routines would access system resources differently, allowing attackers to extract the age group being processed by observing the side channels.

Generally, there are two approaches to mitigate side channels. The first approach is to partition or isolate resources shared between the attackers and the TEEs, for example, to prevent the cloud from swapping out TEE memory while it is running. However, this approach either requires significant engineering costs – necessitating that vendors modify the current hardware designs – or compromises the cloud's flexibility in efficiently managing resources for multiple users. The second, less intrusive approach, is to modify the resource usage patterns generated from the sensitive application, such that they can no longer be used to infer secrets. Unfortunately, many solutions that adopt this require a significant rewrite of the software,

which is often not scalable in terms of engineering and performance costs.

My proposal aims to explore OS-assisted solutions toward making the second approach for side-channel mitigation more practical and performant. The in-TEE OS can control the placement of data within a TEE's memory, and also all its file accesses. This capability enables the OS to transform resource usage patterns, eliminating side channels, all without requiring any modifications to the resource-consuming applications. This idea was only lightly developed during doctoral research, which introduced a minimal OS that transforms all of a TEE's resource accesses, and can only protect simple cloud applications. Applying this idea to a feature-rich OS like Linux, which is widely used in cloud deployments, introduces several new challenges.

First, the sheer complexity of the OS makes it impractical to attempt to protect all resource accesses across both the OS and application code. Not only would such an approach be prohibitively expensive in terms of performance cost, but it would also demand significant engineering effort. A more sensible approach is to apply protection only to resource accesses that can potentially leak secrets. This raises the next challenge: deciding which accesses should be protected. This is nontrivial, especially in complex workloads where multiple resources are accessed, often by interacting applications. To this end, the work on whole-system provenance analysis, led by Thomas Pasquier and Margo Seltzer at UBC, is particularly promising. Whole-system provenance analysis techniques record all resource accesses made by applications within the system. With this data, these techniques can pinpoint the source (or provenance) of any resource about to be accessed. I plan to incorporate these techniques into my work to provide the OS with a principled mechanism to identify exactly at which point a sensitive resource access is about to be made. This information renders the sensitive resource protection efficient; the OS would only modify the sensitive resource accesses about to be made to eliminate the side channels.

Next, even when these decisions are made, applying protections to resource accesses still implies significant changes to how the OS manages its resources. Such changes can easily break compatibility with existing workloads, undermining the deployability advantage of modern TEEs. A way forward is to develop mechanisms that extend OS resource management in a non-intrusive manner. To achieve this, I plan to explore synergies with the work on application-aware memory management by Alexandra Fedorova. These systems provide extension points to the OS, upon which custom resource management policies for specific applications can be implemented. Thus, the OS side-channel protection would only affect the management of selected applications and resources, while leaving the rest to use standard OS management methods.

Practical private data processing. A common scenario for CCC deployments is *private data processing* involving multiple stakeholders, as shown in Table 1. For instance, companies like 23andMe provide personal health data analytics as a cloud service. Its users upload private data to the service's server, which runs a health analytics program on the data and returns the results to the user. The service must handle users' personal data carefully – this is, while mandated by regulations like GDPR, often not trusted by users. A dilemma arises: On the one hand, the user wants to verify that the service is using their data responsibly, for example, never giving it to third parties. On the other hand, to prove this, the service must publicize its data processing to users for verification, which might contain trade secrets.

Stakeholder	Sensitive Asset	Security Interest
Service users	Personal data	Ensure data confidentiality
Service providers	Proprietary data processing	Keep trade secrets confidential

Table 1: Representative private data processing stakeholders, and their assets and interests.

To resolve this *conflict of interests*, existing deployments employ a mediator in a TEE, which is trusted by both service users and providers (Figure 2). The service provider first sends the data processing program to the mediator, trusting it to keep the program secret. Similarly, the users provide personal data

without needing to examine the program. On users' data, the mediator now runs the service provider's program in a way that strictly maintains the confidentiality of users' data. Specifically, the mediator restricts the program from performing any action that it considers dangerous, for instance, communicating with other applications or saving temporary computation results into a file on the disk. Existing mediator designs employ a static and minimal list of allowed operations that the data processing can perform; because of this, many mediator designs fail to support various programs, especially complex ones that access multiple types of resources and those that communicate with other programs.

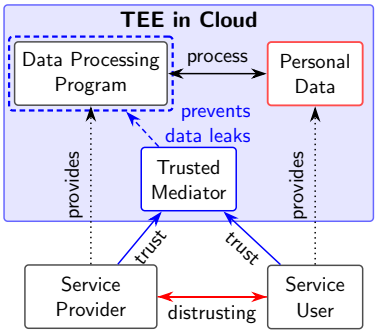

Figure 2: How CCC resolves stakeholders' conflicts.

My proposal looks to improve the usability of existing mediator designs by exploring how the trusted mediator can be incorporated into the in-TEE OS. The rationale is that, since the OS is in charge of resource management for the TEE, it is able to track how the data processing application interacts with sensitive data. With this information, the program's operations are restricted in a "smart" manner – stopping operations only when they may lead to information leaks, but allowing them otherwise. Compared to a static allowlist approach, this dynamic approach would allow the mediator to be compatible with a wider range of compliant programs.

A challenge with this direction is how to reliably detect the propagation of sensitive data across the complex codebase of the OS. To this end, the work on whole-system provenance analysis, led by Thomas Pasquier and Margo Seltzer at UBC, is particularly promising. Whole-system provenance analysis techniques record all resource accesses made by applications within the system. With this data, these techniques can pinpoint the source (or provenance) of any resource about to be accessed. I plan to incorporate these techniques into my work to provide the mediator with a mechanism for determining whether a particular operation could potentially leak sensitive data, allowing the mediator to strictly negate information leaks in a principled manner.

The second challenge is that modern cloud workloads require multiple cooperating machines that interact in a distributed fashion. This means that the trusted mediator must be able to facilitate cross-TEE communications between data processing programs while maintaining data privacy. I plan to tackle this challenge with the help of Mohammad Shahrad from the ECE department. I envision his expertise in designing compliance tools in distributed cloud systems would help in designing the communication between different mediators that must maintain the secrecy of data.

1 Appendix: Guide (REMOVE IN FINAL)

The proposal should be written in clear, non-technical language that allows a non-specialist to comprehend the overall content and importance of the work. Members of the Killam Postdoctoral Fellowships and Prizes Committee are from abroad and range in disciplines, and may not have expertise in your area of study.

Although the fellowship may be used to extend or expand upon doctoral work, it must be made clear that you are not intending to use the award to wrap up a thesis. While it is expected that a postdoctoral fellow will be taking the next step beyond the PhD thesis, you must differentiate clearly between the postdoctoral project and the thesis research. Feel free to include hyperlinks to provide links to additional information.

As applicants must make UBC their base, it is important – particularly for applicants whose primary research materials are elsewhere – to indicate what travel is involved, to where, and for how long. You should describe how you will deal with the remoteness of the primary materials.

The adjudication committee is very interested in "fit" with the selected UBC department or unit and the university's research programs. You must provide information on how your research relates to that of specific campus programs and advisors. If a colloquium is envisioned, a possible title should be proposed. If you visit the classes, please suggest which ones. Since interdisciplinarity is often a valuable dimension (the Killam Trusts declares that a candidate shall not be "a one-sided person"), specific details on proposed interdepartmental connections are welcome.