Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

Noah Starckjohann

Trusted Execution Environments for Privacy-Preserving Statistical Computation in the Cloud

Master's thesis in Information Security
Supervisor: Tjerand Silde
July 2025

NTNU
Norwegian University of
Science and Technology

Noah Starckjohann

Trusted Execution Environments for Privacy-Preserving Statistical Computation in the Cloud

Master's thesis in Information Security
Supervisor: Tjerand Silde
July 2025

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

NTNU
Norwegian University of
Science and Technology

Trusted Execution Environments for Privacy-Preserving Statistical Computation in the Cloud

Noah Starckjohann

July 14, 2025

Acknowledgements

I would like to thank my supervisor, Associate Professor Tjerand Silde at NTNU, for his steady guidance, thoughtful feedback, and patience throughout this thesis. His support was instrumental in helping me navigate both the technical and conceptual challenges of the work.

I am also grateful to Xeni Kristine Dimakos and Li Chun Zhang from Statistics Norway for engaging discussions and for providing practical insights into how this topic intersects with real-world needs in the public sector. Their perspective helped ground the research in its intended context.

This thesis was written while I was an intern at Microsoft, which at times made balancing the workload a challenge. Still, I feel fortunate to have had the opportunity to explore such an exciting and relevant topic while gaining industry experience. The process has deepened my interest in low-level security, a subject I find genuinely fascinating.

Finally, I would like to thank my friends and family for their encouragement, and for reminding me to take breaks when I needed them. Working on this thesis has been both demanding and rewarding, and I am grateful to everyone who supported me along the way.

Abstract

National Statistical Offices (NSOs) are under increasing pressure to modernize how they process sensitive data. As the availability of data from various sources increases, so does the demand for faster and more detailed statistics. To meet these expectations, NSOs must often rely on public cloud infrastructure, which introduces new challenges related to security and privacy. Trusted Execution Environments (TEEs) offer a possible solution by enabling secure computation on potentially untrusted infrastructure.

In this thesis, we investigate the feasibility of using TEEs, particularly Intel SGX, to support privacy-preserving statistical processing in the cloud using data from multiple independent sources. To answer this question, we combine literature analysis with a detailed security model and threat analysis. We then present a system architecture and protocol that supports remote attestation, secure data ingestion, and privacy-preserving record linkage and aggregation.

The results show that TEEs can offer strong confidentiality and integrity guarantees, but that correct use is more difficult than in traditional systems. Development effort increases, and systems can fail silently if assumptions are misunderstood or if the threat model does not match reality. While Intel SGX has known limitations, including side-channel vulnerabilities and exposure to microarchitectural attacks, it remains one of the most robust options for isolating sensitive computation in practice.

For NSOs, deployment involves weighing the trade-offs between enclave design, library OS solutions, and confidential virtual machines. TEEs are not a complete solution, but with the right design decisions and a clear understanding of their limitations, they can support privacy-preserving statistical workflows in a cloud setting.

Sammendrag

Nasjonale statistikkbyråer står overfor økende krav om å modernisere hvordan de behandler sensitive data. Tilgangen på data fra ulike kilder vokser stadig, samtidig som det stilles høyere forventninger til hyppigere og mer detaljert statistikk. For å møte disse behovene benyttes det i økende grad offentlige skytjenester, noe som medfører nye utfordringer knyttet til sikkerhet og personvern. Trusted Execution Environments (TEEs) fremstår som en appellerende løsning, ved å muliggjøre sikker databehandling på potensielt usikker infrastruktur.

I denne oppgaven undersøker vi hvorvidt TEEs, spesielt Intel SGX, kan brukes til personvernbevarende statistisk behandling i skyen med data fra flere uavhengige kilder. For å besvare dette spørsmålet kombiner vi litteraturstudie med en detaljert sikkerhetsmodell og trusselanalyse. Vi presenterer deretter en systemarkitektur og protokoll som støtter ekstern attestasjon, sikker datainnsamling, samt personvernbevarende kobling og aggregering av data.

Funnene viser at TEEs kan gi sterke garantier for konfidensialitet og integritet, men at riktig bruk krever mer innsats enn tradisjonelle løsninger. Utviklingsarbeidet er mer krevende, og systemer kan feile uten varsel dersom forutsetninger misforstås, eller trusselmodellen ikke stemmer overens med virkeligheten. Intel SGX har kjente begrensninger, inkludert sårbarhet for sidekanaler og mikroarkitekturelle angrep, men fremstår fortsatt som et av de mest robuste alternativene for isolering av sensitiv databehandling.

For statistikkbyråer innebærer praktisk bruk å vurdere ulike tilnærminger, som manuell utvikling av enclaver, bruk av bibliotekbaserte operativsystemer, eller konfidensielle virtuelle maskiner. TEEs er ikke en allsidig løsning, men med riktige designvalg og en god forståelse av teknologiske begrensninger, kan de støtte personvernbevarende statistikk i skybaserte miljøer.

Contents

Figures

Tables

Code Listings

Chapter 1

Introduction

1.1 National Statistical Offices

National Statistical Offices (NSOs) have long been the authoritative producers of official statistics, but in today's data-rich society their position is rapidly evolving. Their role is grounded in key principles such as impartiality, scientific soundness, statistical confidentiality and the protection of individuals' privacy, as outlined in international frameworks including the *Fundamental Principles of Official Statistics* [1] and the *European Statistics Code of Practice* [2]. These institutions are responsible for delivering accurate, timely, and policy-relevant information that supports evidence-based decision-making by governments, researchers, and the public. The processing of sensitive personal data is subject to strict legal and ethical obligations. In the European context, the GDPR mandates privacy by design, data minimization, and explicit consent for data use [3]. NSOs must navigate these constraints while ensuring the scientific validity and reproducibility of their outputs.

Traditionally, NSOs have collected census data, surveys and administrative data, approaches characterized by full control over data collection, processing, and dissemination [4]. Under this "closed input–open output" model, raw microdata remained protected within institutional boundaries, while only curated, anonymized outputs were released [5]. This model enabled strong guarantees of confidentiality and methodological transparency, forming the bedrock of public trust in official statistics. However, this approach often resulted in significant delays in data release, reflecting the historical trade-off where achieving high accuracy often came at the expense of timeliness [6].

1.2 Limitations of the Current Statistical Method

Traditional statistical data collection methods, such as surveys and censuses, suffer from significant downsides, especially in today's data-rich environment. Surveys depend on voluntary participation and are increasingly affected by self-selection

bias, where certain demographics are over- or underrepresented. The process of manually completing surveys places a high burden on respondents, often resulting in low response rates and participation fatigue [5].

Even when participation is sufficient, the quality of responses may be compromised by measurement errors and recall bias. Respondents may misreport or forget information, introducing noise into the data. Furthermore, the time lag between data collection, processing, and publication means that insights are rarely available in real time. This delay is particularly problematic when timely information is essential, such as during public health emergencies[1] or rapidly changing socio-economic conditions [7].

These limitations reduce the precision, timeliness, and overall utility of the data that NSOs can provide, and have prompted growing interest in supplementing or replacing traditional approaches with new forms of data acquisition and analysis.

1.3 Opportunities in the New Data Ecosystem

The exponential growth of digital data[2] has led to a proliferation of data producers across the private and public sectors. As a result, NSOs are "one species among many others in a crowded data ecosystem" [8], and no longer exclusive monopolists of statistics. Private corporations and individuals generate vast amounts of data (often referred to as "big data"), creating both competition and opportunities for NSOs. The value of official statistics increasingly depends on integrating non-traditional data sources to produce more timely, granular insights than classical surveys or censuses alone can offer [9].

A central challenge in modern statistics is how to balance the utility of data with the need to protect individual privacy. Utilizing the computing power of hyperscaler datacenters and the vastness of *big data*, can bridge this gap. Doing so in a public cloud setting, where sensitive information may be processed across administrative and organizational boundaries, may have adverse consequences for data privacy. NSOs must adapt both their technical capabilities and institutional practices.

Why is this becoming relevant now? In the past, when multiple organizations wanted to combine their data for analysis, they had little choice but to hand it over to a trusted third party. This party would then control access to the pooled data and facilitate the analysis. But this model has clear downsides, it concentrates sensitive data in one place, requires strong legal and institutional trust, and intro-

[1] In response to COVID-19, governments across Europe and elsewhere deployed mobile applications for contact tracing, infection status verification, and population-level monitoring. While these measures aimed to provide real-time data, they sparked criticism from privacy advocates and researchers who argued that some implementations violated principles of data minimization, consent, or transparency.

[2] According to Statista, 149 zettabytes was generated in 2024 (`https://www.statista.com/statistics/871513/worldwide-data-created/`).

duces significant operational overhead. Today, with advances in hardware-based security, like Trusted Execution Environment, it is becoming possible to collaborate on data without needing to fully trust the infrastructure or the other parties involved.

1.4 Motivation and Problem Statement

The opportunities presented above also introduce new challenges, particularly in the areas of data linkage, privacy, and public trust. The datasets involved in such integration efforts often originate from multiple sources (such as tax, health, or education records) and contain highly sensitive personal information. Their analytical value lies precisely in this sensitivity and granularity. Yet this same characteristic makes them difficult to access, share, or process under existing legal and ethical constraints. This presents a dilemma: *the most useful data also carries the greatest risk to privacy and is subject to the strictest handling requirements*. This tension has been recognized across the official statistics community. NSOs face significant challenges in adapting to non-traditional data, including technical skills, data governance, legal access, and privacy concerns [7].

A major concern arises when these rich datasets are stored or linked centrally. While centralization can simplify analysis and system design, it concentrates risk and increases the likelihood of confidentiality breaches. In cross-institutional or cloud-based scenarios, where data flows across administrative boundaries, these risks become even more pronounced. Relying on cloud-based infrastructures to keep up with the demands of modern data generation entails some privacy and security risks, as the cloud is essentially an untrusted third party. Although it is often considered more secure than traditional on-premise data centers [10, 11], there are several aspects to consider. The cloud service provider employees have access to the physical infrastructure and can interact directly with the hardware, circumventing security mechanisms in software. Even assuming the provider itself can be trusted, the infrastructure is typically shared among several different customers, and security breaches due to vulnerabilities in the computing stack have become an increasingly frequent occurrence.

Traditional Statistical Disclosure Control (SDC) methods have long been used to protect outputs of statistical processing, ensuring that individual records cannot be inferred from published data (de-anonymization attacks) [5]. But in a cloud-enabled, cross-organizational computing environment, input confidentiality is equally important. This leads to a dual challenge: ensuring both *input privacy* and *output privacy* [12]. Traditional approaches to privacy often rely on altering the data itself, either through suppression, generalization, or noise injection. But changing the data comes at a cost. As famously argued by Ohm, “data can either be useful or perfectly anonymous but never both” [13]. This fundamental trade-off limits the utility of anonymization-based approaches and motivates the need for alternative techniques that preserve data fidelity while still ensuring privacy.

1.5 Emerging Solutions

In this thesis, we examine Trusted Execution Environments (TEEs) as a promising Privacy-Enhancing Technology (PET), and a foundational technology in the field of Confidential Computing [14, 15]. They offer a different path: instead of transforming the data to protect it, they aim to protect the computation environment itself. By isolating code and data within hardware-enforced secure enclaves, TEEs allow sensitive analysis to take place without revealing the raw inputs to the underlying system or external observers [16]. When combined with cryptographic protocols and proper system security, TEEs may offer NSOs a path toward scalable, privacy-preserving statistics and data analytics in the cloud.

The CTO of Microsoft Azure, Mark Russinovich, has stated that "we will become a confidential cloud" and that this is the vision for Azure in the coming years [17]. This is also reflected in the projected market size, which is expected to grow significantly [18]. In the 2024 *Gartner Hype Cycle for Compute* [19], Confidential Computing is positioned within the **Innovation Trigger** phase, approaching the **Peak of Inflated Expectations**. This placement reflects growing industry excitement around the technology's potential, despite its relative immaturity. TEEs, as the core enabling mechanism for Confidential Computing, are therefore gaining visibility as a promising but still evolving solution for securing *data in use*. This transitional stage, marked by hype, investment, and emerging adoption, makes TEEs a timely and critical focus for this work. Yet, despite abundant academic research and the growing number of commercial solutions, there is no single, widely-accepted definition of what a TEE actually is [16]. This definitional ambiguity, especially in a maturing and increasingly relied-upon technology, motivated this research into TEEs' security properties and practical boundaries.

1.6 Research Objective and Questions

The goal of this thesis is to investigate whether TEEs can practically and securely enable privacy-preserving statistical computations on sensitive data in untrusted cloud environments.

While privacy-preserving record linkage provides the motivating scenario, this thesis does not aim to advance or evaluate specific record linkage algorithms. Instead, it focuses on how TEEs can secure the processing of sensitive identifiers within such workflows, particularly in contexts where data is contributed by multiple *data owners*[3].

To guide this investigation, we address the following research questions:

- **RQ1:** How can TEEs be architected into a system that enables NSOs to process sensitive data in the cloud with strong confidentiality and integrity

[3] In this thesis, the term *data owner* refers to an entity (e.g., public agency or private organization) that controls access to a dataset and is responsible for deciding how and under what conditions it may be used in collaborative processing.

guarantees?
- **RQ2:** Do existing commercial TEEs provide security assurances sufficient to meet the privacy requirements of official statistics workflows?
- **RQ3:** What practical and technical challenges arise when adopting TEEs for privacy-preserving record linkage, and do they render this approach feasible for real-world NSO deployments?

1.7 Scope

This thesis does not aim to design new linkage algorithms or evaluate their statistical performance. Instead, discussion of record linkage is limited to its architectural and confidentiality implications. Broader methodological or statistical challenges, such as deduplication, match quality, or false positives, fall outside the scope of this work.

We focus on commercial, x86-based[4] TEEs, with particular emphasis on Intel SGX due to its process-level isolation and broad availability on major cloud platforms. Other architectures, such as Intel TDX and AMD SEV, are included for architectural context and contrast, but they are not the primary targets of the system design or security evaluation.

1.8 Related Work

TEEs have been applied to a broad range of privacy-preserving data processing tasks, including machine learning inference, federated analytics, and secure aggregation. Although the goals and trust assumptions vary across domains, many of these systems offer insights into enclave architecture, data flow design, and the practical challenges of deploying TEEs in untrusted environments. Commercial offerings and deployment-focused considerations are discussed separately in Chapter 5.

Several early works explore how TEEs can protect neural network inference by securing both model parameters and user inputs [20–22]. While not directly concerned with data linkage, these systems reveal common challenges in SGX enclave development, including memory limitations, I/O bottlenecks, and the need to partition computation to fit within enclave constraints.

In federated learning settings, TEEs have been used to protect either the aggregation step [23] or the integrity of local model updates [24], typically assuming a symmetric trust model in which each participant trains locally and periodically exchanges updates with a central enclave. While the orchestration model in this thesis differs—favoring centralized linkage over distributed training—the use of enclaves to isolate sensitive logic under partial trust is conceptually relevant. Also related is the notion of "in situ" computation [7], where analysis occurs close to

[4] Specifically 64-bit modes: AMD refers to it as *x86-64*, while Intel calls it *IA-32e*. Other common names include *Intel 64*, *AMD64*, and *x64*.

the data source. However, practical deployments often place heavy operational burdens on data owners, which this thesis explicitly seeks to avoid.

More directly related are systems that apply TEEs to secure data aggregation or record linkage across organizational boundaries. Birgersson et al. [25] propose a decentralized model in which users upload encrypted data and only release decryption keys to an enclave after verifying its integrity through remote attestation. This opt-in design avoids persistent trust in infrastructure but assumes a user-driven participation model with dynamic engagement. He et al. [26] and Liu et al. [27] describe centralized enclave-based linkage protocols in which multiple data owners encrypt their records using a public key provisioned by a single TEE. While these works use the term "multi-party," they assume uniform trust in a single enclave and rely on all participants provisioning secrets to the same component. In contrast, the architecture proposed in this thesis separates provisioning, attestation, and linkage across multiple enclaves, better aligning with asymmetric trust and institutional constraints.

A closely related conceptual framing is offered by Knauth et al. [28], who present TEEs as dynamic brokers of trust in multi-organization workflows. While their work outlines important architectural ideas, it stops short of addressing implementation details or specific applications like linkage or statistical computation.

Outline of the Thesis

The remainder of this thesis is organized as follows: Chapter 2 introduces the notions of Privacy, Record Linkage, and core concepts of PETs. Chapter 3 provides a deep dive into the architecture and principles of TEEs. Chapter 4 outlines the methodology used in this work. Chapter 5 presents a feasibility analysis, followed by the system architecture and protocol design in Chapter 6. Finally, Chapter 7 and Chapter 8 provide a critical discussion and summary of findings.

Chapter 2

Background and Theory

In this chapter, we lay the theoretical foundation for the thesis. We introduce core concepts and technologies that are essential for designing and evaluating TEEs in the context of privacy-preserving statistics. We begin by discussing what privacy entails and why it is challenging to ensure when processing sensitive data. Next, we examine record linkage, a widely used yet sensitive method for NSOs, highlighting both its practical value and potential risks. We then review a range of PETs, demonstrating how TEEs differ fundamentally in their approach by securing the execution environment itself.

2.1 What is Privacy?

Privacy is fundamentally about control over information: the ability of individuals or institutions to determine when, how, and to what extent data about them is shared or used. In legal and policy contexts, this is often described as "informational self-determination". An older but still influential framing comes from "The Right to Privacy" [29], which describes privacy as "the right to be let alone". These formulations emphasize autonomy, consent, and protection from unwanted intrusion.

For NSOs, privacy is not just a philosophical or legal issue, it is central to maintaining public trust and meeting regulatory obligations when processing sensitive personal data. For example, census records can include identifiers such as names, date of birth, and addresses—information that, if mishandled, could have serious consequences for individual privacy. At the same time, these data are critical for producing accurate and timely statistics. This inherent tension makes privacy not only an ethical and legal concern, but also a practical and technical one.

2.1.1 Types of Privacy in Practice

From a technical perspective, privacy can be structured around two key types of protections, which form the foundation for this thesis:

Input Privacy

Input Privacy concerns protecting data during processing so that no intermediaries, not even the computing environment, can access or infer the original values. A relatable analogy is sending a sealed letter: the contents should remain confidential to everyone except the intended recipient. This type of privacy is especially important for NSOs when linking datasets across administrative boundaries or processing data on untrusted cloud infrastructure.

Output Privacy

Output Privacy focuses on limiting what can be inferred about individual data points from the published results of computations. Even if raw data remains secure during processing, careless or insufficiently protected outputs (e.g., tables or statistical releases) can leak sensitive information through reconstruction or differencing attacks. SDC techniques and Differential Privacy are key tools for enforcing output privacy (see Section 2.3.3).

2.1.2 Security as a Prerequisite for Privacy

Security and privacy are closely related but conceptually distinct. Security, commonly defined by confidentiality, integrity, and availability (the *CIA* triad), is necessary for privacy but not sufficient. A system can meet strong security guarantees and still undermine privacy—either by design or through inference. For example, authorized actors may have access to metadata or behavioral patterns that reveal sensitive information. As former NSA director Michael Hayden famously remarked, “We kill people based on metadata” [30], illustrating that security controls alone do not prevent misuse or overreach. Privacy requires additional protections to ensure that secure systems also respect the interests of the individuals whose data is processed.

2.1.3 Perspective of this Thesis

To provide a focused yet practical treatment of privacy in the context of privacy-preserving statistics, this thesis adopts the simplified perspective of input and output privacy. This captures two key concerns for NSOs: protecting sensitive data during processing (input privacy) and ensuring that published statistics do not compromise confidentiality (output privacy). This approach directly informs the evaluation of PETs in later chapters and ensures conceptual consistency throughout the thesis. A key example of this dual challenge arises in record linkage, a process central to modern statistical workflows, which is explored next.

2.2 Record Linkage in Statistics

Integrating data from diverse sources is essential for NSOs to generate comprehensive, timely, and accurate statistics. Census records, administrative data, and surveys each offer unique but incomplete perspectives on a population; no single dataset alone provides a full picture. Linking these datasets allows NSOs to fill information gaps, improve coverage, and produce richer analyses that better support evidence-based policymaking. However, because these sources are collected independently—often with inconsistent formats, varying data quality, and no shared unique identifiers—linking them reliably is both technically and operationally challenging.

The process of identifying records that refer to the same individual or entity across datasets is known as record linkage (or, interchangeably, data matching or entity resolution). This practical task appears in fields ranging from public health to taxation, yet it shares a common challenge: reliance on *quasi-identifiers*—such as personal details or contact information—that are prone to inconsistencies and errors. As a result, matching is inherently uncertain and computationally demanding, especially at scale.

2.2.1 Probabilistic Record Linkage

Early advances in record linkage came from Newcombe et al. [31], who introduced probabilistic linkage: matching based on the likelihood that two records refer to the same entity, rather than requiring exact agreement. Fellegi and Sunter later formalized this framework with a statistical decision model that classifies pairs as matches, non-matches, or potential matches based on likelihood ratios [32]. Their model remains influential in many linkage systems used by statistical agencies today.

These probabilistic methods continue to evolve, with recent research refining techniques for calculating agreement weights, handling missing data, and automating matching decisions [33]. However, despite these advances, privacy concerns around how quasi-identifiers are processed remain largely unaddressed in traditional approaches.

2.2.2 Typical Linkage Process

Most practical linkage systems follow a general structure consisting of the following steps [34]:

- **Pre-processing:** Data is cleaned and standardized to reduce inconsistencies, such as correcting typos or normalizing date formats.
- **Indexing (Blocking):** Candidate pairs are generated by restricting comparisons to records sharing certain features (e.g., same postal code), reducing the otherwise infeasible number of comparisons.

- **Comparison:** Candidate pairs are evaluated attribute-by-attribute using similarity metrics for strings, or absolute differences for numerical fields.
- **Classification:** Pairs are assigned match probabilities or scores and classified by thresholding or probabilistic models.
- **Evaluation:** Quality is assessed through clerical review or statistical checks, since ground truth labels are often unavailable.

This process is central to statistical integration efforts, but it traditionally assumes that identifying information remains accessible in plaintext throughout.

2.2.3 Privacy Risks

As mentioned, when datasets are linked across institutional, administrative, or national boundaries, the privacy risks grow considerably. Quasi-identifiers must often be compared across datasets, yet exposing them, even temporarily, can violate confidentiality agreements or legal obligations. While some agencies attempt to de-identify data before linkage, such efforts are often ad hoc and provide limited protection [35]. Unique combinations of seemingly benign attributes can still enable re-identification.

Traditional SDC methods focus on *output privacy*, trying to ensure that published statistics do not leak individual information. But these methods do not protect data during computation itself. This leaves a critical gap in workflows where sensitive identifiers must be processed in shared or untrusted environments, such as public cloud infrastructure.

Privacy-Preserving Record Linkage (PPRL) techniques attempt to address this by enabling record linkage without revealing raw personal identifiers [36]. These approaches rely on PETs, such as multi-party computation, homomorphic encryption, or private set intersection, to limit exposure during the linkage process.

However, cryptographic PETs often require complex setup, impose significant performance overhead, or assume fully distributed trust. An alternative is to use hardware-based protections to isolate sensitive processing. TEEs, though not always formally categorized as PPRL [37], offer a practical mechanism to enforce confidentiality during linkage by securing the computation environment itself.

The next section introduces key PETs, comparing their properties and trade-offs in the context of privacy-preserving statistical workflows.

2.3 Privacy-Enhancing Technologies

Privacy-Enhancing Technologies (PETs) are tools and methods for protecting sensitive data during collection, processing, and analysis. For NSOs, they are essential both for regulatory compliance and for safely leveraging new, large-scale data sources (see Section 1.1).

As outlined earlier, this thesis frames privacy around two key concepts: *input privacy*, which safeguards raw data during processing, and *output privacy*, which

ensures published results do not compromise individual confidentiality. Maintaining both is difficult, especially in shared or untrusted environments, and requires well-defined threat models and robust technical measures.

2.3.1 Broader Context of PETs

The United Nations identifies PETs[1] as essential to responsible data stewardship [15]. According to the *2023 UN PET Guide*, they enable data use while upholding privacy standards, reducing risks such as discrimination or misuse, and supporting ethical data sharing across institutions.

PETs vary in approach and applicability. Techniques like Differential Privacy (see Section 2.3.3) offer formal guarantees by adding noise, while cryptographic tools such as Multi-Party Computation (see Section 2.3.4) and hardware-based methods like TEEs (see Section 2.3.5) protect data during processing. Each method carries trade-offs in trust assumptions, performance, and integration complexity [14].

Effective system design requires aligning PETs with data flow. Applying protections close to the source helps minimize exposure and supports principles like data minimization and privacy by design, as required by the GDPR.

PETs are already being explored in practice. Initiatives like the *UN PET Lab* support collaborative experimentation among NSOs, showing how PETs can enable joint analysis without compromising confidentiality. Several real-world examples are discussed in Section 5.2.

2.3.2 Homomorphic Encryption

Homomorphic Encryption (HE) is a cryptographic technique that enables computations directly on encrypted data, preserving input privacy throughout the processing pipeline. The core property is that operations on ciphertexts produce encrypted results which, when decrypted, yield the same output as if the operations had been applied to the plaintexts:

$$E(x \star y) = E(x) * E(y), \tag{2.1}$$

where E is the encryption function, $\star$ the plaintext operation, and $*$ its ciphertext equivalent.

This end-to-end encryption model eliminates the need to expose raw data during processing, making HE especially appealing for scenarios where sensitive data must be outsourced to untrusted cloud infrastructure [38]. For NSOs, it offers a way to perform statistical computations—such as counts, means, or regressions—–without ever revealing the underlying identifiers or values [39].

Most modern HE schemes are based on lattice cryptography, and practical use requires expressing computations as arithmetic circuits. Two characteristics are particularly important: the *size* of the circuit (number of gates) and its *depth*

[1] PETs and Privacy-Preserving Technologies (PPTs) are often used interchangeably.

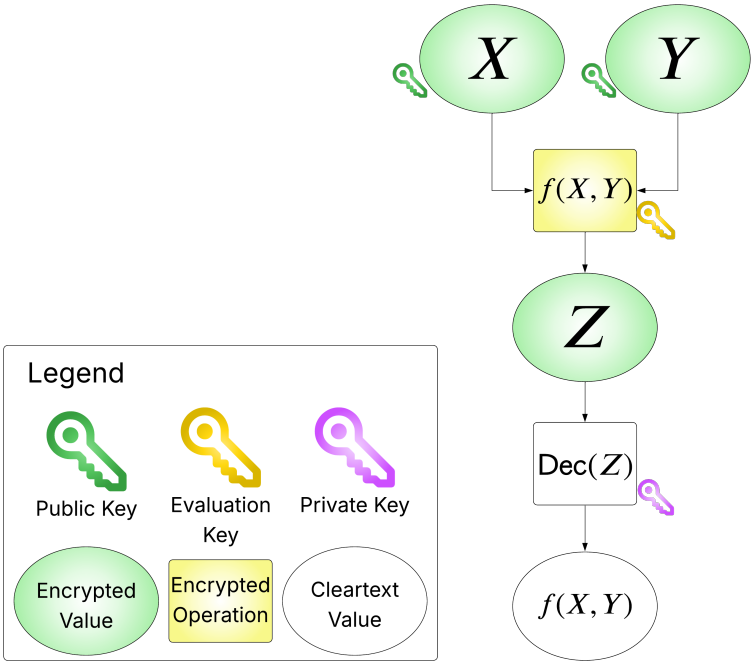


Figure 2.1: Conceptual view of an asymmetric homomorphic operation. Encrypted inputs X and Y are combined homomorphically using an operation f, yielding encrypted output Z. Decrypting Z reveals $f(X, Y)$, equivalent to applying f on plaintexts. A public evaluation key may be needed for certain schemes.

(longest chain of dependent operations). Each operation increases noise in the ciphertext; too much noise can break decryption. Fully Homomorphic Encryption (FHE) allows evaluating circuits of arbitrary depth but incurs heavy computational overheads, while simpler schemes limit depth to remain efficient.

Despite steady progress since Gentry's first FHE construction in 2009 [40], HE remains costly in practice. Computations are orders of magnitude slower than on plaintexts, and ciphertexts are significantly larger. Nonetheless, the ecosystem is maturing with several frameworks and compilers now abstract away cryptographic details [41, 42]. These tools reduce the barrier to entry, particularly for predefined workflows.

While HE may not be suited for complex interactive workflows or large-scale linkage tasks, it remains a promising tool for privacy-preserving analytics on static datasets. It represents a powerful idea: *data can remain encrypted and still be useful*.

2.3.3 Differential Privacy

Differential Privacy (DP) is a formal framework for ensuring *output privacy*: it limits how much information a statistical release reveals about any single individual

in the dataset [43]. Unlike techniques that protect raw data during collection or computation, DP focuses on the release phase, when aggregate results are shared publicly, and aims to prevent inference attacks from compromising individual confidentiality.

DP is especially relevant for NSOs, whose role involves publishing statistics derived from sensitive personal data. The core idea is deceptively simple: *Would the result of this analysis change significantly if one person's data were removed?* If not, the analysis is differentially private. Formally, DP introduces a parameter ε (epsilon), which quantifies the maximum change in the output distribution due to the presence or absence of a single record [44]. Lower ε values imply stronger privacy, as individual influence becomes harder to detect, even with arbitrary auxiliary knowledge.

In practice, DP is implemented by adding calibrated noise to queries such as counts, means, or histograms before releasing the result. The amount of noise depends on both the query's sensitivity and the desired ε, balancing privacy against statistical accuracy. If too much noise is added, results may become useless; if too little, privacy may be compromised.

The *central model* of DP, where a trusted curator holds the raw data and noise is injected before output, aligns well with how NSOs typically operate. This model allows flexible analysis while enforcing privacy at the publication boundary. The UN Handbook recommends DP for protecting microdata and frequently updated aggregates, which are especially vulnerable to differencing or reconstruction attacks [14]. The 2020 U.S. Census marked a major deployment of DP in practice, applying it to redistricting data to counter modern re-identification risks [45]. This adoption drew significant attention but also criticism over reduced accuracy—particularly for small-area statistics or minority populations, where noise can dominate the signal. Such trade-offs remain a key challenge: strong privacy (low ε) often comes at the cost of utility.

DP adoption also requires shifts in practice: designing new workflows, communicating statistical uncertainty, and educating users about privacy guarantees [46, 47]. Nonetheless, its formal guarantees and expanding set of open-source tools[2] make DP a robust candidate for output protection in official statistics.

Combined with techniques for input privacy, DP can serve as a final safeguard, ensuring that even once data leaves secure enclaves, it does not inadvertently expose individuals through aggregate patterns.

2.3.4 Multi-Party Computation

Multi-Party Computation (MPC) allows several parties to compute a joint function over their private inputs without revealing those inputs to each other or any third party [48]. This answers a central question in privacy-preserving statistics: *how*

[2]Examples include Google's Differential Privacy Libraries (`https://github.com/google/differential-privacy`), IBM's DiffPrivLib (`https://github.com/IBM/differential-privacy-library`), and TensorFlow Privacy (`https://github.com/tensorflow/privacy`).

can mutually distrustful parties collaborate without revealing their raw data to each other or a third party? Unlike traditional data aggregation, MPC ensures that participants learn only the final result. This property makes MPC highly relevant in settings where confidentiality is critical, but collaborative analysis is needed [49, 50].

The goal of MPC is to protect input privacy, and given n parties $P_1, \ldots, P_n$ each holding a secret input x_i, they want to compute $y = f(x_1, \ldots, x_n)$ while satisfying:

- *Correctness*: everyone receives the correct output.
- *Privacy*: no extra information is leaked beyond each participant's own input and the shared result.

This replicates the role of a trusted third party without requiring any participant to fully trust others.

The core mechanism is secret sharing: each party splits their input into random shares distributed among the others so that no subset below a certain threshold can reconstruct the secret. Computation then proceeds on these shares, securely evaluating an arithmetic or Boolean circuit. Figure 2.2 illustrates this principle using simple additive sharing.

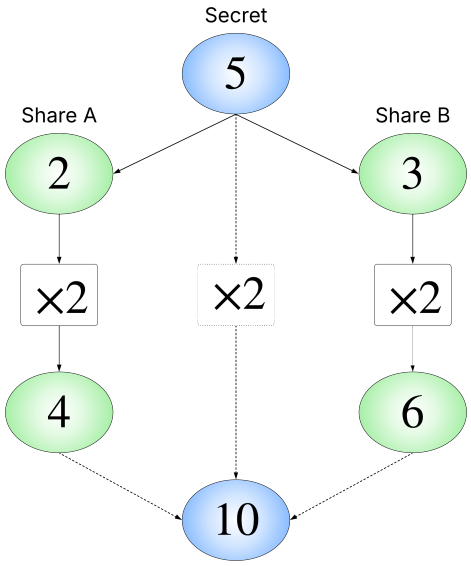


Figure 2.2: Illustration of additive secret sharing. The secret 5 is split into shares 2 and 3, distributed to two parties. Each share is processed locally, and recombining the results yields the correct final value.

Protocols differ in their threat models and guarantees. Some protect against *passive* (honest-but-curious) adversaries, while others also handle *active* (malicious) ones. Security can be *information-theoretic*, resisting even unbounded adversaries, or *computational*, relying on cryptographic hardness [51].

While early MPC protocols were impractical, modern frameworks and optim-

izations have enabled real-world deployments. Frameworks such as CrypTen [52] and Fortified [53] reduce computational and communication overheads, and recent research demonstrates complex workloads like training neural networks or building decision trees over distributed data [54, 55].

Despite these advances, MPC requires significant engineering effort and coordination. As cryptographer Phillip Rogaway noted, "I think if MPC were going to have a profound impact on cryptographic practice, I think it would have had it by now" [56]. This highlights the gap between MPC's technical potential and widespread practical adoption.

2.3.5 Trusted Execution Environments

This section introduces TEEs as a PET. Their architectural principles and security properties are explored in greater detail in Chapter 3.

TEEs are hardware-based[3] technologies that aim to provide *input privacy* by executing code and handling data inside isolated regions called enclaves. These enclaves are protected even if the host operating system or hypervisor[4] is compromised (see Figure 2.4), making TEEs especially relevant for secure computation in untrusted cloud environments.

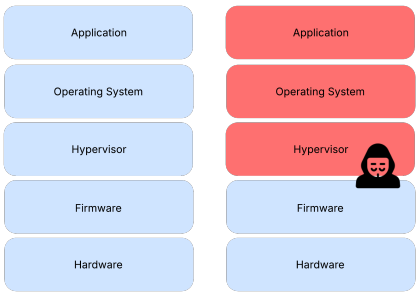


Figure 2.3: Without TEEs: An attacker controlling the hypervisor can access all application memory.

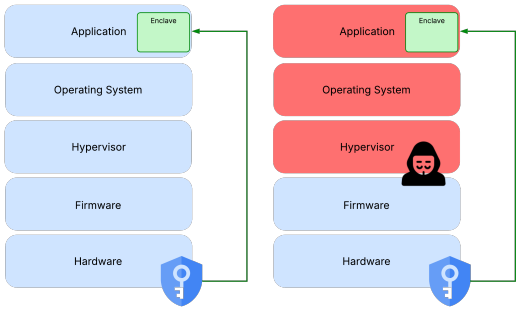


Figure 2.4: With TEEs: Enclaves remain isolated and secure even if the hypervisor is compromised.

Compared to cryptographic PETs like MPC or HE, TEEs offer significant performance benefits by allowing operations on plaintext within the enclave. However, their security relies on hardware assumptions and they do not provide *output privacy*. While TEEs protect data during computation, they do not address what is revealed by the output itself. For comprehensive *privacy* protection, especially in statistical workflows, they should be paired with SDC techniques such as DP[5].

[3] Any mention of TEEs in this thesis refers to hardware-supported TEEs, unless explicitly stated otherwise.

[4] The term hypervisor comes from its role as a "supervisor of supervisors," managing operating systems running in supervisor mode.

[5] DP is considered a modern formal approach within the broader framework of SDC, which also

includes traditional techniques like cell suppression, top-coding, and noise injection.

Chapter 3

Hardware-Assisted Trusted Execution Environments

In this chapter we provide a detailed examination of commercially available hardware-assisted TEEs and the security properties they aim to enforce. We begin by outlining the core architectural principles shared by modern TEEs, followed by a structured review of three major implementations: Intel SGX, Intel TDX, and AMD SEV-SNP. Each is presented with emphasis on threat model, protection guarantees, and relevant technical features, enabling side-by-side comparison of their capabilities and design trade-offs.

Because real-world deployments must contend with the limitations of current TEE hardware, the second half of the chapter shifts focus to attacks. This includes both implementation-level vulnerabilities and broader classes of microarchitectural or side-channel attacks that undermine TEE security. These are systematically categorized to highlight recurring patterns across platforms, and to lay the foundation for the discussion in Chapter 7.

3.1 Security Context

Understanding the security context of TEEs requires a clear view of what they are designed to protect and how trust is established within a system. This section outlines the primary security objectives of TEEs and introduces two foundational concepts, *Trusted Computing Base* and *Root-of -Trust*, that underpin how these protections are realized in practice.

3.1.1 Protecting Data in Use

The confidentiality of data throughout its lifecycle is typically divided into three categories: *data at rest*, *data in transit*, and *data in use* (see Figure 3.1). Well-established defense mechanisms exist for the first two: encryption schemes protect data stored on persistent media (data at rest), while protocols like TLS secure data during transmission over networks (data in transit). However, once data reaches

its destination and is decrypted in memory for processing, it becomes vulnerable to a compromised operating system, hypervisor, or other co-located software.

TEEs address this gap by securing *data in use*, creating hardware-enforced isolated execution environments where data can remain protected even during active computation.

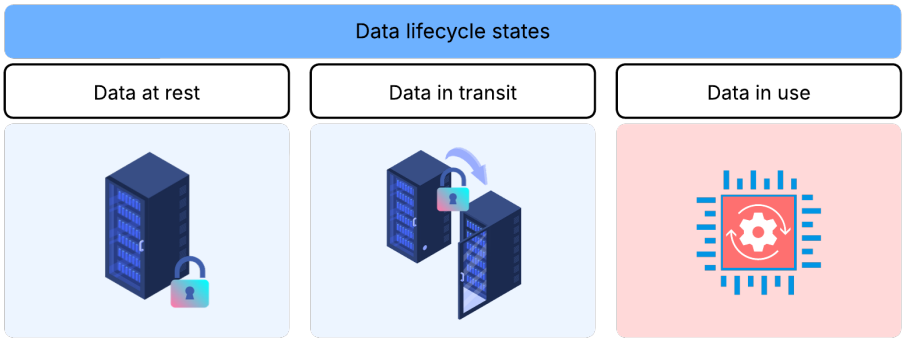


Figure 3.1: Protecting data at rest and in transit is generally considered a solved problem through well-established encryption techniques, securing data in use remains a significant challenge in the data lifecycle.

3.1.2 Threat Model

In order to evaluate an application's security, its threat model must be clearly understood, which is not always a trivial task. Traditionally, the focus has been on protecting the environment, such as the kernel and other processes, from a malicious process. Defenses have therefore been focused on constraining the attacker's capabilities, by means of privilege separation [57], data execution prevention [58], control flow integrity [59] and sandboxing techniques [60], among others [61].

TEEs consider a fundamentally different threat model: one in which the entire surrounding environment poses a potential risk, as both software and hardware components may be compromised and under an adversary's control. A useful mental model is to consider how they differ from sandboxes, which are meant to keep potentially malicious applications from breaking out and causing harm to the system. TEEs are largely focused on the opposite, protecting the application from the system.

Clearly defining the security assumptions in a TEE's design is therefore crucial for evaluating its security posture, understanding which attacks it aims to resist, and identifying appropriate use cases. These assumptions are encapsulated in the system's Trusted Computing Base (TCB), which specifies the minimal set of components that must remain trustworthy for the system to uphold its intended

security guarantees. A deeper look at the concept of the TCB is therefore essential to understanding the security of TEEs and follows in the next section.

3.1.3 Trusted Computing Base

Trust is a subtle and multifaceted concept in security engineering. It is always contextual, inherently asymmetrical, and evolves over time [62]. These aspects make it challenging to define precisely what must be trusted in any given system. This topic is revisited in more depth later in Section 7.3.1, but a foundational understanding is needed here.

The TCB is defined as the set of components that must be trusted to behave correctly for the system's security guarantees to hold [63]. The more components included in the TCB, the greater the potential for vulnerabilities, since every trusted component represents a possible single point of failure. Therefore, trust in the TCB is inherently negative: it denotes the parts of a system that, if compromised, can break security entirely.

In conventional systems, the kernel forms a large part of the TCB and includes numerous complex subsystems, such as the networking stack, storage and filesystem layers, and device drivers. Any of these can be exploited to compromise the entire system. By contrast, TEE designs aim to minimize the TCB by isolating sensitive code and data inside secure enclaves, reducing the amount of trusted code and the attack surface.

A useful framing of this principle is the phrase, "to be able to trust cloud computing, you need to be able to trust it less" [64]. TEEs embody this idea by explicitly designing for scenarios in which the cloud provider or hypervisor might be malicious. Not because this is expected, but to protect against risks like insider threats, rogue administrators, or compromised infrastructure.

Modeling a system through the lens of its TCB provides a systematic way to identify which components must be trusted and which can be assumed under adversary control. For the TEE implementations discussed in this thesis, the processor forms an unavoidable part of the TCB. This is an assumption that, as explored in Section 3.6, can itself become a source of vulnerability.

3.1.4 Root-of-Trust

The Root-of-Trust (RoT) is the component where trust begins in a computer system, and it is a source that can always be trusted to behave in an expected manner [65]. Informally, it anchors security because nothing else can verify or correct it if it fails. Its purpose is to perform security-specific functions, and it may be implemented in hardware, software, or a combination of both. The component itself can be either mutable or immutable, depending on the design goals and threat model. This usually has security implications, as mutable components introduce some level of uncertainty or unpredictability to a system. In some situations, however, it may be necessary to update the RoT, precisely for security reasons, if a

weakness or vulnerability is found. On consumer-based systems, the SEC (Security) phase of the UEFI Platform Initialization process often plays a critical role in establishing the RoT. While the RoT typically includes immutable hardware elements, early firmware components like the SEC phase contribute to extending trust during system startup.

Trusted Platform Modules

Trusted Platform Modules (TPMs) are a type of hardware security module that was introduced to provide a dedicated RoT, with strong physical tampering protection, that the system owner can control [66]. Understanding it can provide useful insights into many of the design decision in TEEs, as many of the concepts were pioneered by TPMs [16].

TPMs broadly consist of two components, secure storage and a cryptographic processor. They are passive devices that other parts of the system can use for security purposes, such as storing keys, generating keys and performing cryptographic operations like signing, attestation, and sealing. TPMs are typically used to support secure boot, platform integrity measurement, and secure credential storage by maintaining internal state that can be queried, but not directly modified. Most modern computers have a TPM, and it is required to run Windows 11[1].

There are several different types of TPMs, which differ in security, but generally provide the same functionality. Discrete TPMs are a standalone hardware unit, separated from the main system. Integrated on the other hand refers to when the TPM is on-chip, and as such "integrated" with the processor. Intel ME (Management Engine) and AMD-SP (Secure processor) implemented what is called firmware TPM. The functionality is implemented by the ME and SP respectively, and not a physically separate unit. Current TEE solutions generally rely on different architectural components, but the fundamental principles remain the same.

TEE technologies typically redefine the RoT, to ensure safe and secure execution, beyond what traditional systems have been able to establish. This is a core aspect of TEEs, and the security guarantees depend on the RoT being in a benign state, operating as expected. There are mainly two different ways this trust can be established, statically or dynamically.

Static Root of Trust for Measurement

A Static Root of Trust for Measurement (SRTM) establishes trust in a system's boot sequence by starting from an immutable Core Root of Trust for Measurement (CRTM), typically located in the platform's boot ROM. The CRTM measures each successive component in the boot chain, storing cryptographic hashes in the TPM's Platform Configuration Registers (PCRs). This process extends a measurement

[1] `https://blogs.windows.com/windows-insider/2021/08/27/update-on-windows-11-minimum-system-requirements-and-the-pc-health-check-app/`

chain where each stage verifies and records the integrity of the next, creating an auditable history of the boot process [67].

Some TEEs rely on SRTM for the secure initialization of lower layers. Secure Boot, Trusted Boot, and technologies like Intel Boot Guard use SRTM to ensure that firmware and early OS components are unmodified and trustworthy before launching. For example, Boot Guard enforces signatures on BIOS code, preventing untrusted firmware from undermining TEE isolation guarantees [68]. AMD-SEV is an example of a commercial TEE that relies on a SRTM [16].

The static nature of SRTM means trust depends on the entire boot chain: a compromise of early components, like BIOS or System Management Mode (SMM), invalidates later trust assurances and can expose TEEs to attacks. This risk is well known on x86 platforms, where attacks on firmware have historically allowed adversaries to bypass or tamper with trusted boot measurements [69].

Dynamic Root of Trust for Measurement

A Dynamic Root of Trust for Measurement (DRTM) enables a platform to establish a RoT at runtime, independently of the initial boot chain. Unlike SRTM, which relies on a static measurement chain from the earliest boot stage, DRTM allows launching a new, measured execution environment even if the system's earlier state cannot be trusted. This is done by invoking a *late launch* instruction, such as Intel's `GETSEC[SENTER]` on TXT-enabled systems, which triggers the CPU to measure and securely initialize a small piece of trusted code called the Measured Launch Environment (MLE) [68].

DRTM is especially relevant for TEEs because it allows creating isolated execution environments on demand, without depending on the entire boot process being secure. This is valuable in scenarios where TEEs must operate on shared or cloud hardware, or where the platform's firmware integrity cannot be fully verified. By performing measurements dynamically, DRTM ensures the TEE starts in a known good state, enabling secure attestation and protected execution even if the system's prior software stack was compromised. Both Intel SGX and TDX use a DRTM [16].

A key advantage of DRTM over SRTM is that it reduces the size of the TCB needed to establish trust. Only the CPU, chipset, and MLE need to be trusted for the measurement, not the entire boot firmware. However, DRTM-based TEEs still depend on hardware features to prevent tampering during the dynamic launch process, and runtime attacks remain a threat if they occur after the trusted environment is established.

3.2 Core Principles of TEEs

There is currently no universally accepted definition of what constitutes a TEE, nor a fixed standard for the guarantees such environments must provide. This ambiguity has led to inconsistent use of the term and varying interpretations in both

academia and industry [67, 70]. Nevertheless, researchers have converged on a common set of core properties that are widely adopted across most commercial and academic TEE implementations. These include verifiable launch (attestation), run-time isolation, and secure storage, often accompanied by trusted input/output [16].

Building on the foundations laid by the RoT mechanisms discussed previously, this section provides an overview of these core principles, focusing on their goals, mechanisms, and implications for system security. The aim is to distill the fundamental security properties that TEEs strive to provide, drawing from systematization efforts [70, 71] and refined conceptual models [67]. Similar to those works, enclaves, trust domains, and secure encrypted VMs will all be referred to collectively as *enclaves* in the following sections.

3.2.1 Verifiable Launch and Attestation

Verifiable launch combines two core mechanisms: measurement of the initial integrity of the execution environment [72], and attestation. Together, these mechanisms establish trust in an enclave's execution state and integrity, enabling the secure provisioning of sensitive data. Informally, verifiable launch proves (to either a local or remote entity) that the software and underlying platform are in an expected, uncompromised state.

The process relies on a RTM, which serves as the anchor for cryptographically measuring the system state, including the enclave's code, configuration, and platform status [71, 73]. These measurements are then used to generate a signed report that can be sent to a remote verifier (remote attestation) or consumed by another enclave or a local verifier (local attestation).

As described in Section 3.1.4, the RTM can be established dynamically or statically using DRTM or SRTM mechanisms, respectively. From this trusted anchor, a chain of trust is formed by measuring each component before execution. Measurements typically consist of cryptographic hashes at page granularity, which can be stored in TPM PCRs, in CPU-protected memory regions, or fused directly into on-chip secure elements depending on the platform [16]. Recent work highlights the growing complexity of attestation workflows in cloud-based confidential computing, especially in virtualized or multi-tenant environments [74].

Remote attestation is essential for enabling external parties to validate the integrity of an enclave before provisioning secrets or sensitive data, ensuring that only trusted and verified code can access them. By verifying the enclave's measured state, remote parties can confidently establish secure channels and provide confidential information without risking exposure to compromised environments.

3.2.2 Run-time Isolation

Run-time isolation refers to the protection of code and data during execution. TEEs enforce strict separation between the trusted execution context and all other system components, including the operating system, hypervisor, and other user

applications. There are different mechanisms for achieving this, each with their own strengths and weaknesses. Separation in time, *temporal*, means resources are only accessible to a single component at a time. Separation in space, spatial, means dedicating a partition of the resource to the component. Typically a strategy combining the two is used, referred to as spatio-temporal partitioning [16, 71].

Enforcements of this separation can broadly be divided into two categories, logical or cryptographic. Logical involves using access control mechanisms to prohibit unauthorized access to a resource. This means an adversary should not be able to view or modify data at all. Cryptographic enforcement takes a different approach, and will often allow an adversary to access the data. Security in this case hinges on the fact that data is encrypted, rendering the plaintext inaccessible to entities that do possess the corresponding key. The integrity is protected using cryptographic Message Authentication Codes (MAC), appended to the data, and can contain freshness information to prevent replay attacks.

CPU and memory isolation strategies will use one or more of the mechanisms described above, though some approaches are more prominent than others. For instance, achieving CPU isolation spatially, such as reserving an execution core exclusively for enclaves, limits resource utilization and is considered too costly. Memory isolation is a crucial, yet challenging aspect of run-time isolation. It is not limited to off-chip memory such as DRAM, but also encapsulates on-chip microarchitectural structures. Most modern systems employ virtual memory, and rely on page tables for correct translations to physical addresses. This data structure is particularly difficult to protect, given the range of attacks an attacker can preform. The isolation strategies and enforcement mechanisms consequently vary depending on the TEE's threat model and performance considerations. Many microarchitectural structures in modern processors are shared across isolation boundaries to maximize performance, which opens the door to side-channel leakage. This has been demonstrated through the numerous attacks targeting these low-level components (see Section 3.6). As a result, microarchitectural protections in TEEs have become an area of increasing focus, though this remains an active and evolving field of research.

3.2.3 Trusted I/O

Ensuring trusted I/O requires both securing the communication channel to the peripheral, referred to as a *trusted path*, and ensuring that the peripheral itself enforces confidentiality and integrity, which is known as a *trusted device architecture*. The purpose of a trusted path is to maintain confidentiality and integrity for the enclave's accesses to the device [16]. When implemented using logical mechanisms, the path is protected against a wide variety of adversaries, including those with control over the boot firmware, system software and peripherals. However, it cannot protect against an attack on the bus (physical interconnect), which requires cryptographic isolation.

The second component of trusted I/O, *trusted device architecture*, typically

relies on temporal partitioning to isolate access to shared peripherals using secure context switches. When multiple enclaves require access to the same device concurrently, spatial partitioning alone may be impractical or insufficient due to hardware limitations. In such cases, a combination of spatio-temporal partitioning and logical enforcement mechanisms is employed. For stronger guarantees, cryptographic enforcement can be applied directly to device-side memory resources, enabling secure binding of data to enclave identities and mitigating risks from compromised device firmware or shared buffers [16].

3.2.4 Secure Storage

Secure storage allows an enclave to persist sensitive data across sessions while ensuring that it remains confidential and tamper-evident. This is commonly achieved using a process called *sealing*, where data is encrypted and authenticated using keys that are unique to the enclave or its developer identity (depending on the sealing key policy). The opposite process, decrypting the data, is naturally called *unsealing*. These keys are typically derived from a hardware RoT and bound to enclave-specific measurements [67].

The actual mechanisms for secure storage are often underdocumented or completely omitted in many TEE design specifications [16], despite there being available architectural primitives for implementing it.

3.3 Intel SGX

Intel Software Guard Extensions (SGX) introduced the notion of secure enclaves to provide integrity and confidentiality guarantees to sensitive applications running in ring 3 (user mode). An enclave is constructed within a user mode application's virtual address space, with hardware mechanisms to protect it from unauthorized access [75]. Intel x86 platforms that have support for SGX provide such isolated execution environments by adding a set of extensions to the architecture and a new CPU mode called enclave mode. This functionality is exposed to developers through new instructions, some of which are available to unprivileged code (Ring 3) [76]. The usage of these closely resembles that of systems calls, with arguments provided in general purpose registers. At a high level, they allow code and data to be incrementally loaded into a dedicated DRAM region, while contentiously generating a corresponding cryptographic measurement [77]. This measurement serves as proof of what is actually loaded into memory, and can subsequently be provided to a third party during the attestation process. Hardware-enforced access controls determine what is permitted when executing in enclave mode and provide protection against other software and enclaves.

The main mechanisms of SGX will be described below, but the bulk of its functionality is implemented in microcode [78, 79]. A variant that Intel calls XuCode [80] is used to deliver parts of it, utilizing a special execution mode. Figure 3.2 shows the relationship between x86 instructions, microcode and XuCode

in the instruction decode part of the processor's execution pipeline. The details of how this works, and what XuCode Mode entails, are not (well-)documented. An investigation into this is out of scope for this paper, but is worth noting since it is technically code that is in the TCB.

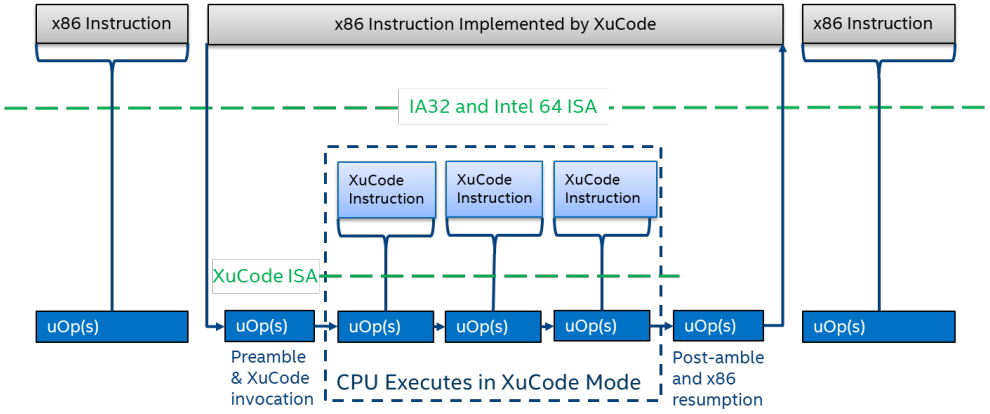


Figure 3.2: Showing how XuCode is used to implement SGX instructions. Figure from [80]

SGX significantly reduces the TCB, as neither the OS, hypervisor, BIOS nor SMM is trusted (see Figure 3.3). Application secrets are protected, even when the entire platform is compromised. Generally, the processor internals are trusted. There is, however, an assumption that there are no side-channels, neither in hardware nor software. Additionally, enclave code is assumed to be correct, and any cryptographic primitives used must be secure. To achieve this security model, SGX utilizes several new data structures. While some will be introduced to illustrate how SGX implements its core TEE primitives, an exhaustive description will not be provided.

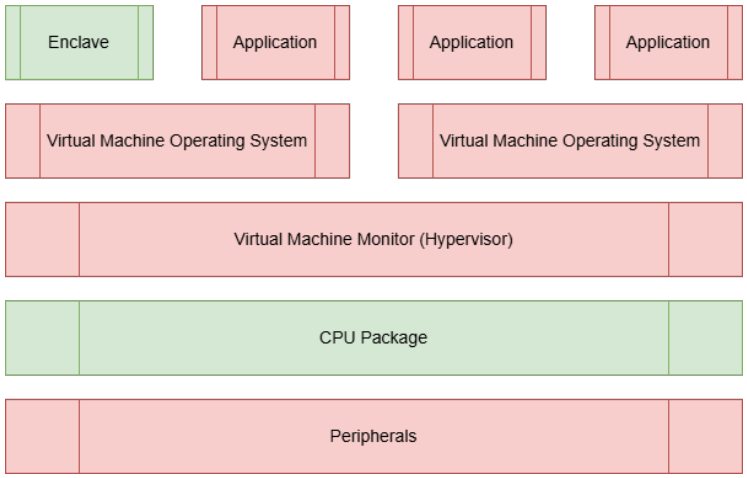


Figure 3.3: A high level overview of the Intel SGX TCB.

3.3.1 Run-time Isolation

SGX enclaves reside within the address space of an unprivileged host application. Each enclave receives dedicated private memory in the form of 4KB pages within the Enclave Page Cache (EPC). The EPC is a reserved region of physical memory called Processor Reserved Memory (PRM), isolated from other software by hardware mechanisms [76, 81]. Firmware (BIOS or UEFI) configures this isolated memory region by programming dedicated range registers. An important enclave parameter is the *Enclave Linear Address Range*, defining the contiguous virtual address range occupied by the enclave. All enclave memory accesses must fall within this range, ensuring strict spatial isolation within the host application's address space and preventing accidental or malicious memory references outside enclave boundaries. The size of PRM is limited and varies between SGX versions: SGXv1 supports regions up to 128MB [82, 83], while SGXv2 (targeting server-grade Xeon processors) expands this limitation to the range of 128GB to 1TB depending on the processor model [84, 85].

While the PRM provides physical isolation, the underlying DRAM is external to the CPU package and therefore considered untrusted. To address threats from attackers who can observe or tamper with DRAM contents, SGX employs a hardware component known as the Memory Encryption Engine (MEE). The MEE encrypts data flowing between the CPU and DRAM and maintains integrity and freshness through an integrity tree (Merkle-Tree) and associated MAC (see Figure 3.4). The root of this integrity tree is securely stored in the processor's on-die SRAM, ensuring data integrity even in the face of physical memory attacks [77]. With the increased EPC memory that SGX2 enables, it now uses AES-XTS encryption without Merkle-tree integrity checks, weakening this integrity protection [84] (see Section 7.1.3 for security implications)

Understanding how EPC pages are managed requires examining the enclave life cycle, starting from enclave creation. The `ECREATE` instruction, initiated by system software (kernel or hypervisor), begins enclave creation by establishing an SGX Enclave Control Structure (SECS). The SECS resides in protected memory within the EPC, inaccessible to any software component outside the enclave. It contains the enclave's identity and attributes, including measurement registers (`MRENCLAVE`, `MRSIGNER`), configuration flags (e.g., debug mode), and feature bits. These fields are critical for enforcing enclave policies and for attestation integrity checks. Although system software is responsible for allocating resources, including memory, security-critical metadata about EPC pages is managed by the hardware through the Enclave Page Cache Map (EPCM). The EPCM records page-specific security attributes, such as the enclave ID, virtual address, permissions, and page type. On memory accesses, the processor consults both the OS-managed page tables and the EPCM. The EPCM ensures the validity of memory accesses by verifying enclave mode execution, page ownership, virtual-to-physical address mappings, and access permissions. The EPCM is stored internally within processor microarchitecture, invisible and inaccessible to software, ensuring security even

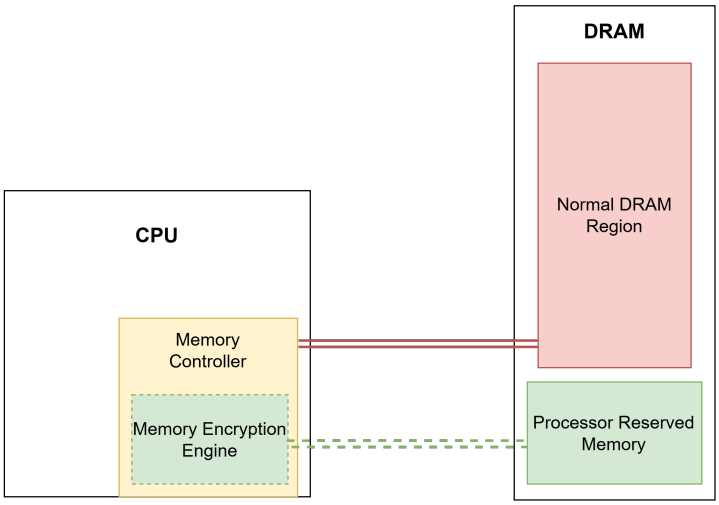


Figure 3.4: Simplified illustration of the Memory Encryption Engine (MEE). All data traffic between the processor and PRM is encrypted and integrity-protected by the MEE.

against malicious OS page table manipulations [75]. See Figure 3.5 for a visual representation of how the EPCM relates to the other structures mentioned above.

SGX's run-time isolation uses a combination of spatio-temporal partitioning and logical enforcement [16]: EPC pages are allocated exclusively to enclaves over time (temporal) and kept spatially isolated in PRM, while hardware mechanisms like EPCM validations enforce correct access permissions (logical). Meanwhile, the TCB relies on purely spatial partitioning within the processor (microarchitecture), and cryptographic isolation[2] (via the MME) protects enclave memory against physical attacks on DRAM.

Following enclave creation with `ECREATE`, the `EADD` instruction loads initial code and data into the enclave's EPC[3]. `EADD` uses a `PAGEINFO` structure specifying source and destination addresses, along with a reference to the SECS structure identifying the enclave owning the new EPC page. Security permissions and page attributes are defined in the accompanying SECINFO structure. At this stage, enclave memory is initialized without secrets, remaining visible to system software. Subsequent security-critical operations, such as secret provisioning and attestation, occur later in the enclave lifecycle and are discussed in Section 3.3.2.

Execution transitions into the enclave occur through `EENTER`, which switches the processor into enclave mode, while `EEXIT` safely transfers control back to untrusted code, maintaining isolation during synchronous transitions [86]. A crit-

[2]With SGX2, cryptographic isolation weakened: AES-XTS encryption was adopted without Merkle-tree integrity protection to scale EPC sizes, reducing guarantees of memory integrity [84].

[3]In SGX2, additional instructions such as `EAUG` support dynamically adding EPC pages at runtime. See Section 7.1.3 for security implications.

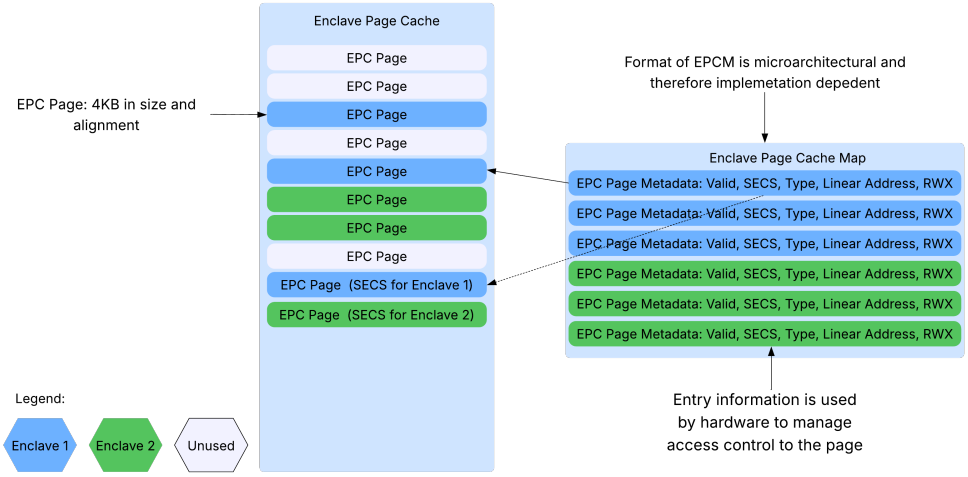


Figure 3.5: Relationship between EPC, EPCM, and SECS structures. The EPCM tracks the metadata and permissions for each EPC page, enforcing secure enclave memory access.

ical aspect of SGX's run-time isolation is the management of asynchronous events through Asynchronous Enclave eXits (AEX). When an interrupt or exception occurs, the processor performs an AEX, pausing enclave execution and securely saving CPU state into the State Save Area (SSA) within enclave memory. The enclave can later resume using `ERESUME`, restoring state from the SSA. In newer SGX versions, the optional AEX-Notify mechanism allows enclaves to receive notifications upon AEX events before resuming, enabling the enclave runtime to react to frequent interrupts and mitigate attacks like single-stepping by inspecting or handling these events directly [87].

3.3.2 Attestation

SGX provides enclaves with the ability to prove their authenticity and correctness to a third party. A key component of this mechanisms is that it utilizes trusted hardware to establish a DRTM, enabling the platform to securely measure and attest to the enclave's state independently of the system's boot history. During the enclave creation process (see Section 3.3.1), a secure cryptographic log is recorded [88]. This log contains measurements of the enclave's initial state. The `EADD` instruction measures the virtual address space layout, the order and relative position of pages, and their security properties. The `EEXTEND` instructions measures the memory contents, including the code, data, stack and heap [81]. After the construction of an enclave's initial state, as described in Section 3.3.1, the `EINIT` instruction completes the enclave setup. It is then no longer possible to add content to the enclave, and the log if finalized. In SGX terminology, this final 256-bit digest is referred to as `MRENCLAVE` ("Enclave Identity"), a field of the

SECS data structure [81]. Given the components that contribute to the hash, it serves as a unique identifier for an enclave. If the code, data, or memory layout were to be slightly altered, the hash would change completely, in accordance with the cryptographic property of bit-level diffusion. This prevents a malicious enclave loader from intentionally setting values that trigger unexpected behavior. As such, the choice of what to include in `MRENCLAVE` is a critical security decision, although excessive restrictions can limit enclave deployment flexibility.

`MRENCLAVE` is not the only identity associated with an enclave, it also has a certificate-based identity made up of `MRSIGNER`, a product identifier and a version number. A Singing Identity (sometimes referred to as Sealing Identity), `MRSIGNER`, represents the identity of the enclave author (developer) [89]. This is the entity that originally built and cryptographically signed the enclave prior to distribution [88]. If `MRENCLAVE` provides a way to gain confidence in an enclave's contents, `MRSIGNER` can satisfy the concern regarding who built the enclave.

An enclave can perform either local or remote attestation. Local involves attesting to another target enclave on the same platform using symmetric encryption, while remote extends this to entities outside the platform using asymmetric encryption. It is important to note that remote attestation is necessary in order to accurately determine that the enclave is running on a real SGX platform [90].

Local attestation uses the `EREPORT` instruction to create a signed attestation report (REPORT). This instruction provides oracle access to the target's Report Key, which it otherwise would not be able to obtain [91]. This key is only known to the target enclave, and the `EREPORT` instruction. It contains `MRENCLAVE`, `MRSIGNER`, enclave attributes, hardware TCB, optional user data, and a MAC tag [88]. The MAC is used by the recipient of the REPORT and is created using the *Report Key*. This key is only known to the target enclave, and the `EREPORT` instruction. The report is then sent to the target enclave, which will use the `EGETKEY` instruction to obtain the Report Key, and verify the content of the report. The target enclave then reproduces the steps performed by the originating enclave, so that both enclaves can be assured they are executing on the same platform. The optional user data in the `REPORT` can be used to create a secure channel using Diffie-Hellman Key exchange. Consequently, the result of local attestation can be a shared symmetric key between two enclaves on the same platform.

Due to the inherit challenges of proving execution on real hardware, remote attestation requires a additional steps and is significantly more complex [91]. A pair of privileged architectural enclaves created by Intel have special access to a set of cryptographic keys. Since they are created by Intel, they are assumed to be secure. These enclaves, in turn, acquire another set of keys. A special architectural enclave called Quoting Enclave (QE) is given raw access to these keys [91]. Since a REPORT is only locally verifiable, a Report Key is not employed during remote attestation. Instead, the QE signs the report using a Quote Signing Key (QSK) certified by Intel and accompanied by a certificate chain rooted in Intel's attestation CA. This turns the REPORT into what Intel refer to as a quote, essentially a remotely verifiably `REPORT`. The remote party verifies that the quote was signed by a

legitimate SGX platform. It is through this process that a relying party can enforce security requirements on the client system where the enclave is executing.

Intel has offered a remote attestation service based on Enhanced Privacy ID (EPID). This followed a client-based model, aiming to enable remote authentication and attestation of hardware while preserving privacy [92]. EPID used a group signature scheme that allowed revocation of group membership, which was especially useful if a device or its cryptographic keys became compromised. As of April 2025, the original EPID-based attestation service offered by Intel is reaching its end-of-life[4], and Intel now recommends the DCAP model for remote attestation. DCAP is discussed in the protocol design in Section 6.2.1.

3.3.3 Trusted I/O

Intel SGX inherently lacks built-in support for Trusted I/O, meaning enclaves cannot directly establish secure communication channels with external peripherals like keyboards, displays, or network interfaces. This limitation arises because SGX primarily focuses on protecting enclave memory and computation from compromised software environments, rather than handling the complexities associated with secure device interactions. Consequently, enclaves must rely on potentially compromised operating system drivers and peripheral interfaces for I/O operations, significantly limiting their overall security guarantees.

Several research projects have proposed supplemental solutions to bridge this gap. SGXIO introduces a trusted path architecture that combines enclaves with a small trusted hypervisor. It establishes a cryptographically protected channel between enclaves and I/O devices, mediated by a trusted intermediary that isolates device access from untrusted OS components [93]. Aurora similarly employs a hypervisor-based approach, providing secure I/O paths specifically designed for client-side peripherals. Aurora emphasizes user-centric security, aiming for broad compatibility with existing operating systems and minimal performance overhead [94].

Device-specific approaches also exist. For instance, BASTION-SGX [95] uses architectural enhancements to create secure Bluetooth communication paths, ensuring cryptographic protection and logical isolation from potentially compromised system software. Meanwhile, Rakis [96] introduces optimized secure I/O primitives designed to enhance the performance of trusted enclave interactions with external devices, focusing specifically on efficiency across enclave and OS boundaries.

Despite these advancements, Intel SGX itself does not incorporate generic Trusted I/O mechanisms. Existing solutions introduce additional complexity and potential performance trade-offs, highlighting an ongoing challenge in integrating secure and efficient trusted I/O within SGX's native architecture.

[4] https://www.intel.com/content/www/us/en/developer/archive/tools/sgx-attestation-service-utilizing-epid.html

3.3.4 Secure Storage

Intel SGX's run-time isolation provides confidentiality and integrity only to executing enclaves; once the enclave terminates, all internal data is lost. To enable persistence of enclave state or secrets across enclave instances or platform reboots, SGX introduces a secure storage mechanism known as *sealing*.

Sealing encrypts and integrity-protects enclave data using a platform-specific Sealing Key derived from a Root Sealing Key, also known as Fuse Key, which is generated and stored securely within the CPU during manufacturing [81, 88]. Thus, sealed data can only be unsealed by enclaves running on the same physical platform, ensuring platform-bound confidentiality.

The sealing operation uses the `EGETKEY` instruction, which derives the Sealing Key based on a chosen sealing policy. Two policies are available, determining which enclaves can subsequently unseal and access the stored data. Sealing based on the enclave's identity (`MRENCLAVE`) restricts data access strictly to enclave instances with exactly the same code, memory layout, and state, as even minor changes alter the cryptographic measurement. This offers maximum security but complicates software updates and compatibility across different enclave versions.

Alternatively, sealing based on the signing identity (`MRSIGNER`) allows enclaves signed by the same author, potentially across multiple versions or even different applications, to share sealed data [88]. In this scenario, the derived key additionally binds to the enclave's Product ID and Security Version Number (SVN), allowing controlled data migration and version management. While less restrictive, this approach significantly simplifies updates and interoperability among related enclaves.

SGX enclaves thus face a critical trade-off when choosing the sealing policy, balancing security and flexibility. Enclaves are responsible for choosing and securely implementing encryption and integrity protection schemes using keys derived via `EGETKEY`, as Intel SGX itself provides only the underlying cryptographic primitives rather than complete secure storage solutions [81, 88].

3.4 Intel TDX

Intel Trust Domain Extensions (TDX) is a TEE that enables secure virtual machines, known as Trust Domains (TDs). TDX extends the Intel x86 architecture, offering hardware-backed isolation and integrity for VMs without trusting the hypervisor or OS. The TCB includes the processor itself, related virtualization technologies (VT-x, TME, and SGX), and Intel-signed software modules.

TDX introduces a new architectural execution mode, Secure Arbitration Mode (SEAM), extending Virtual Machine Extensions (VMX) instructions built upon Intel Virtualization Technology (VT-x)[97]. SEAM provides two execution modes: SEAM VMX root mode and SEAM VMX non-root mode. Transitions into and out of SEAM root mode are managed by special instructions, `SEAMCALL` and `SEAMRET`, from VMX root mode. Trust is anchored in silicon-fused public keys verified through

MCHECK instructions and Intel Alias Checking Trusted Module[5] [98, 99]. Intel also provides the SEAM Loader (SEAMLDR), responsible for securely loading and managing TDX modules. The SEAMLDR securely verifies and loads TDX modules into protected SEAM memory regions, mitigating attacks through strict access controls [100].

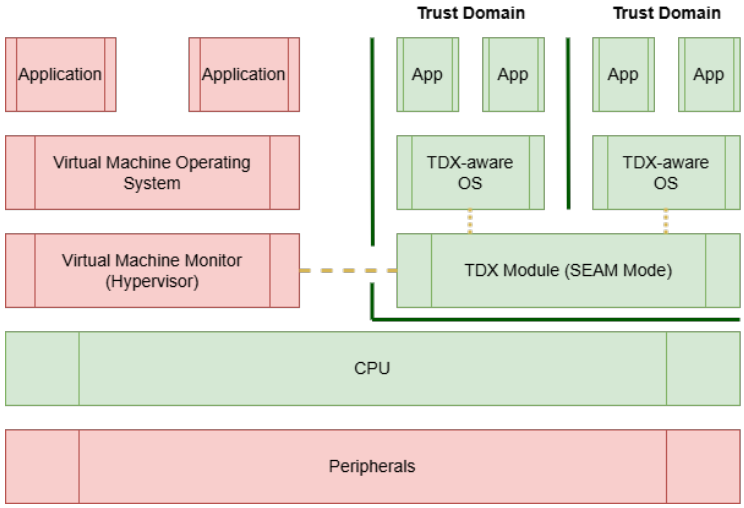


Figure 3.6: A high level overview of the Intel TDX TCB.

3.4.1 Run-time Isolation

Intel TDX utilizes Intel Virtualization Technology (VT-x) and SEAM to provide robust isolation for TDs. Under normal circumstances, a hypervisor runs in VMX root mode while guest VMs run in VMX non-root mode. TDX expands upon this architecture by adding two new VMX modes: SEAM VMX root and SEAM VMX non-root mode. This separation is enforced by hardware-based access controls and cryptographic isolation, preventing unauthorized access by other TDs, privileged software, and attackers with physical memory access [101].

Each TD employs two separate Extended Page Tables (EPTs): A secure-EPT mapping private encrypted memory and a shared EPT mapping shared unencrypted memory and Memory-Mapped I/O (MMIO). Address translation for private memory is securely managed by the Intel-signed TDX module firmware [98].

Within SEAM root mode, execution is restricted to specific memory regions defined by SEAM Range Registers. These are divided into two protected subranges: `MODULE_RANGE`, hosting the TDX Module, and `P_SEAMLDR_RANGE`, hosting the Persistent SEAMLDR (P-SEAMLDR). The Non-Persistent SEAMLDR (NP-SEAMLDR), an authenticated code module, initializes these ranges and securely loads the P-

[5]Vulnerabilities have been found in these: `https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-01111.html`

SEAMLDR. A vulnerability in NP-SEAMLDR could severely compromise the security foundation of TDX, underscoring its critical role [100].

The P-SEAMLDR securely manages TDX module installation, verifying cryptographic signatures, setting up data regions, stacks, and SEAM transfer VMCS structures for logical processors. It also records module identities in CPU measurement registers before returning control to the Virtual Machine Monitor[6] (VMM) via the `SEAMRET` instruction [100].

TDX memory management involves two essential structures: Trust Domain Memory Regions and Physical Address Metadata Tables. These track page ownership, type, and attributes, preventing unauthorized allocation or modification. Each TD's metadata structures, including the control structure, virtual processor state, and secure-EPT, are encrypted using TD-specific private keys to maintain confidentiality and integrity throughout the TD lifecycle [98, 101].

Memory isolation and integrity are primarily ensured by Intel's Total Memory Encryption Multi-Key (TME-MK) engine. TME-MK provides unique encryption keys per TD, encrypting private VM memory contents to protect against unauthorized access, including attackers with physical DRAM access. A special *shared bit* within the Guest Physical Address (GPA) toggled by the TDX-aware guest OS kernel indicates the EPT to be used for resolving Host Physical Addresses (HPAs) [102, 103].

TDX supports two distinct memory integrity mechanisms: Cryptographic Integrity (Ci) and Logical Integrity (Li). *Ci* employs a 28-bit Hashed MAC stored alongside encrypted memory content in DRAM, enabling detection of memory tampering or Rowhammer attacks. *Li* tracks unauthorized writes using a TD Owner bit, marking memory segments as poisoned to prevent compromised data from propagating within the TD or TDX Module [101].

Context switching between the hypervisor and TDX Module is securely managed using dedicated VMCS structures: SEAM Transfer VMCS (for hypervisor interactions) and TD Transfer VMCS (for TD interactions). These structures isolate sensitive state transitions, ensuring that only authorized entities control execution contexts. Secure runtime memory exchanges are facilitated by dynamically mapped keyhole regions, maintaining isolation and protecting sensitive data during interactions between the TDX Module and external entities [98].

3.4.2 Attestation

Similar to SGX, TDX has support for remote attestation of TDs. The attestation process begins when the host VMM creates a TD using a sequence of TDX module calls. At this point, the TDX module initializes the Measurement Register of the Trust Domain (MRTD), a cryptographic hash that accumulates the initial state of the TD [98]. The VMM populates the TD's memory by invoking the hypercall `TDH.MEM.PAGE.ADD`, which registers the GPA and memory type for each page. The contents of these pages are then measured in 256-byte chunks using

[6] Often referred to interchangeably as a hypervisor.

TDH.MR.EXTEND, which updates the MRTD digest. Once all pages have been added, the VMM calls TDH.MR.FINALIZE to finalize the measurement. At this point, the MRTD becomes immutable. This measurement does not include the second-level address translation structures, such as EPT, used to translate guest addresses to host physical addresses, which has implications for completeness of the trust chain [104].

To support dynamic measurements during execution, TDX provides four Runtime Measurement Registers (RTMRs), which are initialized to zero and can be extended by the TD using the guest-callable TDG.MR.RTMR.EXTEND instruction [101]. These registers function similarly to TPM PCRs, allowing for internal state to be tracked over time, such as through measured boot processes inside the TD itself. The VMM has no visibility into or control over RTMRs.

To initiate attestation, the TD generates a TDREPORT using the TDG.MR.REPORT interface. This report includes the MRTD and RTMR values, TD configuration attributes, and platform-level security information including CPU SVN and the TDX module's own measurement hash and version [98]. The report also includes a 64-byte REPORTDATA field, supplied by the caller, which typically contains a nonce or public key hash. The entire TDREPORT is authenticated using a MAC derived from a hardware-protected key only available to the TDX module and other local trust anchors. This MAC ensures the authenticity of the report but renders it verifiable only on the same platform.

Remote attestation reuses the SGX quote generation infrastructure: the TDREPORT is verified by a local SGX QE and signed to produce a Quote [101]. While the infrastructure for quote generation and verification from SGX is reused, the underlying trust assumptions differ. Notably, the TDX attestation flow depends on the integrity of the TDX module, SEAM runtime, and system firmware (including SEAMLDR), which are outside the scope of the measured MRTD. These components are assumed to be part of the platform's RoT and are verified only indirectly via the TCB versioning metadata [99]. Moreover, unlike SGX, TDX does not implement a memory integrity tree, meaning that the confidentiality of TD memory is protected through encryption, but its freshness or integrity cannot be independently verified—potentially enabling replay or rollback attacks in certain scenarios [104].

Despite these limitations, TDX attestation provides a practical and scalable mechanism for establishing trust in VM-based confidential workloads. The decoupling of measurement (by the TDX module) and signing (by the SGX QE) allows integration with existing data center attestation frameworks, while maintaining cryptographic assurance over the TD's launch and configuration state.

3.4.3 Trusted I/O

Intel introduced *TDX Connect* as an extension to the TDX and VT-d frameworks to support secure, high-throughput I/O for Trust Domains (TDs) [105]. It enables TDs to communicate directly with trusted peripheral devices, referred to as *TEE-*

IO devices, while ensuring that device access remains cryptographically isolated from the rest of the system. See Figure 3.7 for a visual representation.

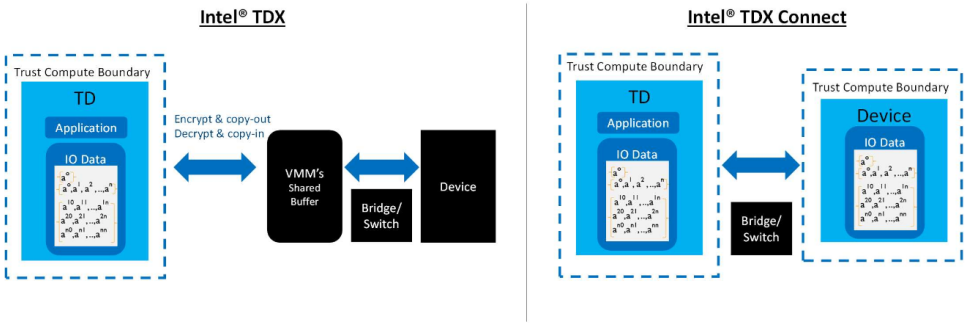


Figure 3.7: Comparing I/O Virtualization with TDX vs. TDX Connect. Figure from [105].

The architecture addresses three core goals: mutual attestation between the TD and the device, integrity and confidentiality of the PCIe data path, and verifiable lifecycle control of assigned device interfaces [106].

To meet these goals, TDX Connect incorporates several protocols and hardware components:

- *TDISP (TEE Device Interface Security Protocol)* governs the authentication, attestation, and lifecycle control of TEE-IO devices.
- *SPDM (Security Protocol and Data Model)* is used to exchange signed measurements and establish encrypted sessions.
- These exchanges are transmitted using *PCIe DOE (Data Object Exchange)*, which must be supported by both the device and platform.

Runtime data protection is enforced through *PCIe IDE (Integrity and Data Encryption)*, specifically via selective IDE streams. These streams are cryptographically bound to a TD and its assigned device, protecting Transaction Layer Packets (TLPs) from tampering or interception. PCIe switches between the TD and device are not trusted and excluded from the TCB [107].

Provisioning is handled by a special service component called the *TEE-IO Provisioning Agent (TPA)*, which is itself a Service TD. The TPA performs device attestation and facilitates secure interface handoff. The TD must then explicitly accept the device, referred to as the *TEE-IO Device Interface (TDI)*, after verifying its identity and expected measurements.

Once established, access to the device is strictly controlled:

- **MMIO** pages are mapped into the TD's Secure-EPT, and transactions are tagged with a private *KeyID* that sets the trusted 'T-bit' in each TLP. Devices validate the T-bit and stream metadata before accepting the operation.
- **DMA** access works similarly, with the TDI using IDE-protected packets. Physical pages are restricted via DMA remapping tables, enforced by the IOMMU

and managed by the TDX module [108].

This model ensures that only explicitly accepted devices can interact with TD memory through well-defined, attested, and encrypted channels. However, its security depends on proper protocol implementation, correct device behavior, and strict lifecycle enforcement by the platform.

3.4.4 Secure Storage

TDX does not provide a native mechanism for sealing data or deriving persistent cryptographic keys within the TD. Unlike Intel SGX, which includes architectural support for enclave-specific sealing keys tied to measurement values, TDX deliberately omits such functionality from the TD's execution context [98, 101]. This design decision simplifies the TDX threat model and avoids expanding the hardware TCB, but it also means that secure storage must be handled externally.

If a TD requires the ability to store data persistently, it must establish a secure communication channel with a trusted external service, such as a Service TD or provisioning agent, that can handle encryption, key management, and access control. These services may use attestation to authenticate the TD and enforce sealing policies based on its identity and configuration. However, this is not part of the TDX specification itself, and there is no architectural interface that allows a TD to derive persistent keys from its MRTD, RTMRs, or other measurement registers [99].

As a result, secure storage in TDX is not guaranteed by the architecture alone, and must be provided as part of a higher-layer trusted runtime or orchestration infrastructure. This increases the reliance on external components for confidentiality and integrity of persistent data, which may have implications for trust assumptions in multi-tenant or adversarial environments.

3.5 AMD SEV

Secure Encrypted Virtualization (SEV) is AMD's hardware-backed approach to isolating virtual machines (VMs) from a privileged hypervisor. Unlike traditional virtualization, where the hypervisor retains full visibility into guest memory and register state, SEV encrypts guest DRAM using a per-VM AES key managed by the AMD Security Processor[7] (AMD-SP, or just SP) [109, 110]. This design aims to prevent even a compromised host from inspecting VM contents, shifting trust away from software and toward a hardware root-of-trust.

The SP is a dedicated ARM Cortex-A5 core embedded in the CPU die. It generates and provisions encryption keys, enforces memory ownership via hardware-tagged ASIDs, and produces attestation reports rooted in AMD's signing infrastructure [111]. Crucially, these functions are implemented in SP firmware, stored in off-chip SPI flash, and run in isolation from the x86 cores. While this makes SEV

[7] Also referred to as Platform Security Processor (PSP) in various documentation.

more transparent to guest software than enclave-based models like Intel SGX, it also increases reliance on vendor-controlled firmware for security.

SEV was first introduced with the Naples generation of AMD EPYC processors in 2017. Its initial threat model aimed to protect against an untrusted or even malicious hypervisor. Under this assumption, SEV encrypted memory to block passive reads from the host, but did not protect CPU register state, memory integrity, or I/O paths. SEV-ES (Encrypted State), introduced later in 2017, addressed register state exposure during VMEXITs [112], but side-channel and replay attacks persisted [113, 114].

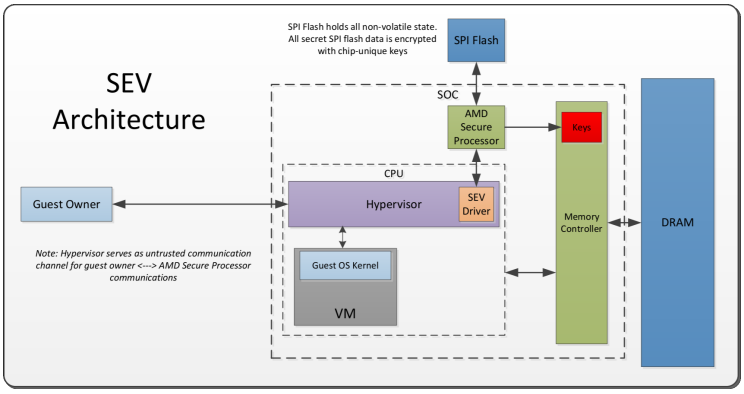


Figure 3.8: Overview of AMD SEV architecture. Figure from [109].

SEV-SNP (Secure Nested Paging), introduced with the Milan generation in 2021, reflects a more robust adversarial model in which the hypervisor is fully untrusted. It augments SEV-ES with integrity protection through a hardware-enforced Reverse Map Table (RMP), VM privilege levels (VMPLs), and fine-grained control over which operations require guest approval [115, 116]. These features aim to prevent memory remapping, stale state re-use, and unauthorized device access. The platform also supports runtime attestation and secure interrupt delivery, enabling VMs to verify their own execution environment beyond launch time [117, 118].

SEV and its variants differ from enclave-based TEEs in that they isolate entire guest operating systems and their applications within a confidential VM. This broader attack surface makes SEV more versatile in deployment, but also exposes it to a wider range of hypervisor- and system-level attacks. Nevertheless, in many scenarios—particularly in the cloud—SEV-SNP raises the baseline security posture by reducing host visibility, enforcing guest-controlled policy, and enabling measured provisioning.

The following sections examine how SEV enforces runtime isolation, handles attestation, and extends its trust boundary to cover I/O and persistent state. While SEV-SNP addresses many limitations of earlier designs, its security properties depend not only on hardware enforcement but also on firmware correctness, key management, and how tightly the surrounding infrastructure adheres to its threat

model.

3.5.1 Run-time Isolation

At runtime, AMD SEV enforces cryptographic isolation between virtual machines by encrypting guest memory with a key derived and managed by the SP. Each VM is associated with a unique 128-bit AES key, and all DRAM accesses are tagged with an Address Space Identifier (ASID) that determines which key the memory controller uses [110]. The encryption is applied transparently in hardware and introduces only modest performance overhead. This mechanism prevents a compromised hypervisor from reading the contents of guest memory, as unauthorized access yields ciphertext encrypted under an unknown key.

SEV's isolation guarantees are spatial and temporal. Spatially, memory is partitioned between VMs and the hypervisor via ASIDs and controlled C-bits in page table entries. Temporally, access to secrets is implicitly bounded to the duration of the VM session, as encryption keys are never exposed to software and are wiped on shutdown. However, SEV's original design did not protect CPU register state or enforce memory integrity, making it susceptible to control flow manipulation, ciphertext reuse, and register exfiltration [109, 114].

SEV-ES (Encrypted State) was introduced to harden `VMEXIT` behavior by encrypting and sealing register contents. This prevents a hypervisor from observing or modifying CPU state during context switches. The Virtual Machine Control Block (VMCB) is split into a control area visible to the hypervisor and a save area encrypted by the CPU [112]. To support necessary hypervisor-guest interactions (e.g., for emulating instructions or I/O), SEV-ES adds the #VC (VMM Communication) exception. The guest handles these events using a shared Guest-Hypervisor Communication Block (GHCB), enabling selective exposure of state. This architecture reduces attack surface, but requires hypervisor and guest kernel cooperation [115].

SEV-SNP (Secure Nested Paging) expands SEV-ES to support a stronger threat model: a fully malicious hypervisor with arbitrary control over physical memory mappings and I/O. To counteract memory replay and remapping attacks [114, 116], SEV-SNP introduces the Reverse Map Table (RMP), a hardware structure that binds physical pages to specific guest contexts. This prevents unauthorized aliasing or reuse of encrypted pages. SEV-SNP also adds cryptographic integrity verification for memory contents, enabling detection of tampering before decryption. These measures shift SEV's isolation model from best-effort secrecy to enforceable data provenance.

Additionally, SEV-SNP exposes a firmware interface for runtime attestation. Guests can verify their execution state beyond launch, including current firmware version, memory layout, and security policy settings [118]. This capability supports stronger provisioning workflows where secrets or data are injected only after dynamic verification. It also enables tighter coupling between trust decisions and observed runtime state.

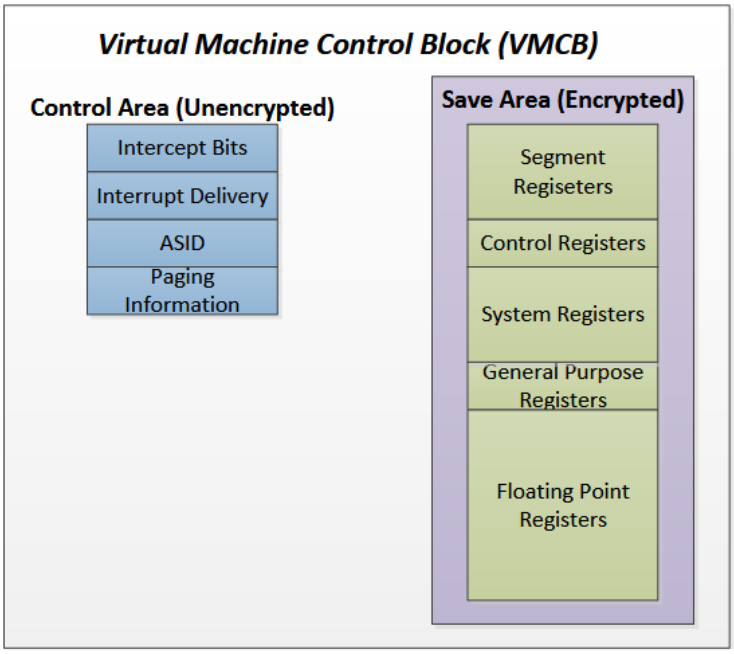


Figure 3.9: SEV-ES divides the VMCB into an encrypted section and an unecrypted section. Figure from [112].

3.5.2 Attestation

Attestation in AMD SEV provides a remote party with cryptographic evidence that a VM has been launched on genuine AMD hardware and is running in a known good state. The mechanism is designed to establish trust directly between a VM owner and the SP, bypassing the hypervisor and host operating system. This is particularly important in cloud deployments, where the hypervisor may not be trusted.

When an SEV-enabled VM is launched, the SP computes a measurement of the VM's initial state—specifically, a cryptographic hash over its memory contents and associated metadata, such as launch policy flags [111]. The result is bundled into an attestation report, which includes the measurement, the platform's firmware version numbers, and policy settings that define which host actions (e.g., migration, debugging) are permitted. This report is then signed using a hardware-rooted attestation key, allowing the VM owner to verify both the platform's identity and the integrity of the VM at launch.

SEV's attestation chain is anchored in AMD's SRTM. Each processor includes a unique Chip Endorsement Key (CEK), derived from a fused-on secret and used for signing attestation reports. The CEK's public component is certified by AMD through an intermediate key, the AMD Signing Key (ASK), which in turn is signed by the AMD Root Key (ARK) [115]. This forms a certificate chain that allows VM owners to validate that an attestation report originates from a genuine AMD plat-

form. Prior to launch, the SP also generates a Platform Diffie-Hellman (PDH) keypair, which is used by the VM owner to securely provision secrets. Only if the attested measurement and policy match the expected values will the SP release the decrypted secret to the guest, ensuring that sensitive material (e.g., statistical datasets or decryption keys) is only accessible inside the correct VM state [111].

Originally, this attestation process occurred only at launch, with the SP signing a static snapshot of the VM's configuration. SEV-SNP expands this model by allowing the guest itself to request fresh attestation reports at runtime via a secure firmware interface [115, 118]. This supports dynamic trust decisions and enables re-attestation in response to events such as configuration changes, firmware updates, or provisioning of new workloads. Runtime reports include the current TCB version, measurement, and other configuration state, and are signed using the same certificate chain rooted in the CEK.

Despite its cryptographic structure, the security of the attestation flow depends heavily on the integrity and correctness of SP firmware. Earlier versions of SEV were shown to allow key extraction and firmware rollback, undermining attestation and enabling full compromise of supposedly protected guests [111]. These issues have since been mitigated through firmware updates and architectural changes in SEV-SNP, but the broader lesson remains: static trust in vendor firmware can be fragile. Attestation in SEV-SNP provides strong guarantees under normal conditions, but the robustness of the trust chain ultimately hinges on secure firmware provisioning and a non-subvertible SP.

3.5.3 Trusted I/O

While SEV encrypts guest memory and protects CPU state, it does not by default protect data once it leaves the memory subsystem.. In traditional SEV deployments, data written to or read from devices—such as storage drives or network interfaces—passes through shared system buses and DMA-capable devices in plaintext. This exposes a blind spot in SEV's isolation model: I/O channels remain under hypervisor control and may be monitored or tampered with. To address this, AMD introduced SEV-TIO (Trusted I/O), an architectural extension that encrypts and authenticates I/O traffic at the hardware level [119].

SEV-TIO builds on the PCIe Integrity and Data Encryption (IDE) protocol, which enables secure, end-to-end encrypted communication between the CPU's root complex and compliant devices. This channel is established using keys provisioned by the SP and bound to the current VM context. As a result, data transferred over the PCIe bus—whether to a network card, NVMe drive, or accelerator—is both confidential and integrity-protected. Even if the hypervisor or DMA engine attempts to intercept the transfer, the data remains encrypted and unauthenticated. Decryption is only possible by the designated device, which must also support SEV-TIO and PCIe IDE [120].

From a security model perspective, SEV-TIO explicitly extends the VM's trust boundary to include a specific hardware device. This requires not only I/O hard-

ware support, but also that the guest trusts the device firmware and its attestation state. To facilitate this, the SP mediates the establishment of device-level trust using SPDM-based attestation, ensuring that only approved endpoints are included in the encrypted I/O path [119].

In practice, SEV-TIO is not yet widely adopted. It requires platform firmware support, compatible PCIe devices, and hypervisor integration to manage I/O mappings and trust relationships. However, for deployments involving sensitive workloads, such as encrypted analytics pipelines or confidential database queries, SEV-TIO can provide meaningful protection against threats that persist even with full memory and CPU isolation. By eliminating plaintext exposure on the I/O path, it closes one of the last remaining gaps in the SEV threat model.

3.5.4 Secure Storage

While AMD SEV does not implement a dedicated secure storage subsystem, it supports encrypted persistence of VM state, such as disk images, memory snapshots, and cryptographic secrets, through its memory encryption and attestation mechanisms.

Secrets can be provisioned into a VM only after successful attestation. Typically, they are encrypted to a key derived from a Diffie-Hellman exchange with the SP, and released only if the VM's measurement and policy match expected values [111]. This enables sealed storage: encrypted secrets can survive reboots or migration, but are only accessible within the correct VM configuration.

SEV-SNP strengthens this model by adding memory integrity enforcement. Encryption keys remain within the SP, and the RMP ensures that pages restored from snapshot are correctly bound to the originating VM [115]. This enables confidential snapshotting and suspend/resume without exposing memory contents to the hypervisor.

Still, secure snapshotting is not foolproof. Rollback and aliasing attacks remain possible without additional protections such as monotonic counters or trusted metadata. Formal analyses highlight these residual risks and stress the need for careful snapshot handling and migration policy enforcement [116].

When used correctly, SEV enables encrypted backups and VM mobility without relying on guest cooperation or exposing secrets to the host. The security of such storage, however, ultimately hinges on trust in the SP's enforcement logic and firmware integrity.

3.6 Analysis of Attacks on TEEs

TEEs are often presented as offering strong security guarantees even in the presence of a compromised operating system or hypervisor. However, over the past decade, a wide range of attacks have demonstrated that these guarantees can be undermined through microarchitectural, physical, or software-based vulnerabilities. Some of these weaknesses can be addressed with software patches or

microcode updates, but many arise from fundamental hardware design choices that are impractical or impossible to change after manufacturing.

Although attacks could have been presented alongside each individual TEE, they are consolidated into this section because many exploit shared underlying principles—such as cache side channels or transient execution, which transcend specific TEE implementations. Covering every known attack is infeasible, as new variants and refinements are published frequently, and small modifications can significantly affect applicability across different hardware revisions. As noted by Weiser et al., "it is increasingly difficult to track the applicability of various attack techniques across the SGX design landscape" [121], reflecting both the pace of attack evolution and the complexity introduced by frequent hardware updates. For more comprehensive reviews, readers are referred to surveys such as [70, 121].

A particular emphasis is placed on cache side channels in this section. This is not only because they directly threaten confidentiality, but also because they form the foundation of many other attack techniques, including transient execution attacks like Spectre, Meltdown, and their variants. Cache-based side channels can be leveraged as a building block in attacks leaking secrets, even in the absence of software bugs [121], underscoring their central role in TEE exploitation.

Analyzing these attacks is important for two reasons. First, it demonstrates that TEEs do not offer absolute security and that their trust boundaries can be circumvented, which is critical when designing privacy-preserving protocols that rely on enclaves. Second, it highlights that confidentiality guarantees provided by TEEs must always be contextualized within evolving microarchitectural threats, emphasizing the need for defense-in-depth and careful protocol design. Many of the attacks covered affect multiple TEE platform, as is often the case [70].

This section provides an intuition for where and how things can go wrong, focusing on representative attacks rather than exhaustive coverage. Categories of attacks are introduced to facilitate a clearer, comparative understanding of the threats that remain relevant for practical deployments of TEEs in privacy-sensitive applications.

3.6.1 Side-Channel Attacks

Side-channel attacks exploit indirect information leakage through effects like timing, power consumption, or shared hardware state, enabling attackers to infer secrets without violating explicit architectural protections. In a more traditional sense, they can be thought of as information disclosure attacks, but are distinct in that information is not observed directly. While these leakages do not directly expose data architecturally, they often allow recovery of secrets with partial accuracy, some error, or more generally a success probability $p < 1$. In the context of TEE, side channels are particularly insidious: they undermine the confidentiality guarantees that enclaves or secure VMs are meant to provide, even in adversary-controlled systems. A prominent example is the class of controlled-channel at-

tacks [122], where an untrusted operating system leverages page faults to infer enclave memory access patterns at page granularity.

A key driver of progress in researching these attacks has been the development of powerful single-stepping frameworks and techniques that enable instruction-level control over TEEs. Early work [123] showed how a malicious operating system could use timer interrupts and cache probes to extract secrets from SGX enclaves with high spatial and temporal resolution—without access to hardware debug features. This inspired more systematic tools such as SGX-Step [124, 125], SEV-Step [126], and TDXdown [127], which generalize such stepping attacks into reusable frameworks. These tools have enabled systematic exploration of microarchitectural leakages and the creation of granular exploits, including attacks capable of recovering cryptographic keys or precise control flow information. As TDXdown recently demonstrated [127], even new TEEs like Intel TDX that implement countermeasures remain susceptible to such fine-grained interrupt-driven attacks.

Understanding and building an intuition for side-channels is thus fundamental: not only do they represent the most widely exploited class of TEE attacks to date [70, 121], but cache side-channels in particular serve as building blocks for many other attack vectors, including transient execution exploits such as Spectre and Meltdown. The following sections examine representative side-channel attacks, organized by the leakage method.

Cache Side-Channels

Modern processors rely on caches to reduce memory access latency by keeping frequently accessed data in small, fast SRAM close to the CPU. Caches are structured hierarchically: L1 and L2 caches are private to each core, while the Last-Level Cache (LLC, typically L3) is shared among all cores. The inclusiveness of the LLC—where data in L1/L2 must also reside in the LLC—means eviction from L3 removes it from private caches as well. This property allows attackers to infer or manipulate cache contents of other cores, including in virtualized environments [128]. See Figure 3.10 for a visual illustration of the cache hierarch.

While caches are essential for performance, they also inadvertently leak information. Specifically, access times differ between cache hits and misses, and timing these differences can reveal whether a memory location was used by a victim, leaking sensitive data. Most modern caches are set-associative, meaning memory blocks map to specific sets within the cache (see Figure 3.11). Exploiting caches this way requires understanding the caches are organized and their cache set mapping, eviction strategies, and addressing——details which vary by cache level. For example, L1 caches often use virtually indexed, physically tagged addressing, while L2 and LLC are physically indexed and tagged [129].

The mapping of virtual or physical addresses to cache sets is crucial for finding *congruent addresses*—memory locations mapping to the same cache set. The LLC is further divided into slices (typically matching the number of cores), and a hash function involving physical address bits determines which slice a cache line is

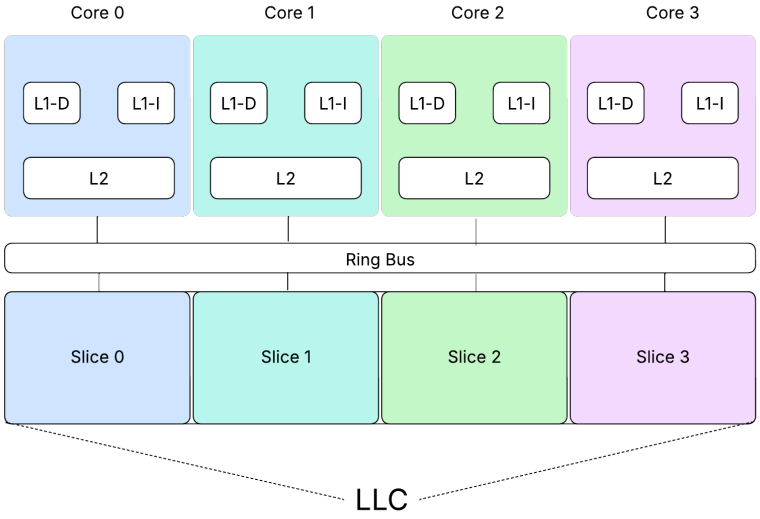


Figure 3.10: Example cache hierarchy of a generic Intel-based 4-core processor. Each core has private L1 instruction, L1 data, and L2 caches. All cores are connected via a ring bus to a shared unified LLC, implemented as slices distributed across the die. *All slices are shared and accessible by all cores*.

mapped to [130].

Cache side-channels have constituted a significant threat since their emergence in 2005 [131], evolving in practicality and sophistication over time. Representative cache attack techniques include:

The **Evict+Time** [132] method measures the execution time of a victim's computation immediately after the attacker has evicted specific cache sets. A noticeable slowdown suggests the victim accessed those locations. While coarse-grained and susceptible to noise, it can detect victim usage at cache-set granularity and does not require shared memory. Similarly, **Flush+Reload** [128] flushes shared memory lines using the `CLFLUSH` instruction, waits for victim execution, then times reloads; fast reloads indicate victim access. This technique provides instruction-level resolution but necessitates shared pages, rendering it ineffective against enclaves lacking shared memory [133]. In scenarios where shared memory is unavailable, **Prime+Probe** [134] is often preferred. The attacker fills specific cache sets, allows the victim to run, then probes to detect evictions. This approach functions across CPUs and VMs but requires identifying congruent addresses, often through reverse-engineering physical address mappings [130]. Automating the discovery of cache-based leakages, **Cache Template Attacks** [129] involve a profiling phase to map victim events to cache sets, followed by an exploitation phase that monitors these sets to infer secrets.

These methods have proven devastating for TEEs. CacheZoom [133] demonstrated that a malicious OS can interrupt enclaves with high frequency, achiev-

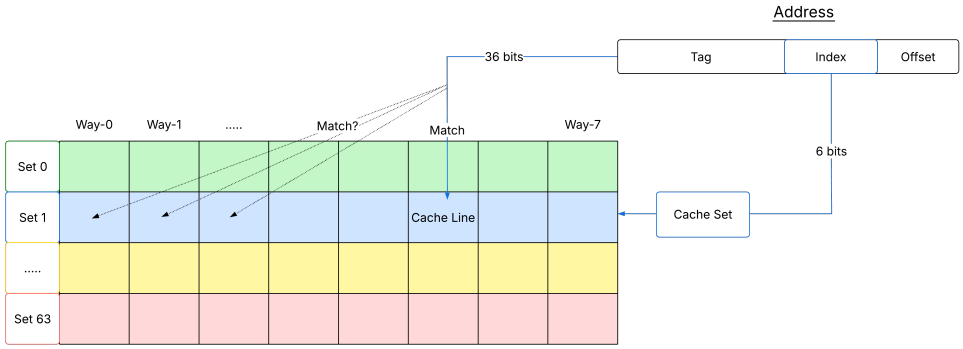


Figure 3.11: 64-bit address lookup into a set-associative cache with 64 sets, 8 ways, and 64-byte lines, resulting in a 32KB cache (64x8x64=32,768 bytes), typical for an L1 data cache on x86.

ing near-instruction-level L1 monitoring and recovering AES keys from SGX enclaves in as few as 10 measurements. Software Grand Exposure [135] extended these attacks to non-cryptographic privacy-sensitive workloads (e.g., genome processing), illustrating that SGX's side-channel vulnerability affects more than just cryptography. CacheOut [136] and SGAxe [137] advanced these techniques by forcibly evicting victim data from caches into leaky buffers, enabling full enclave memory extraction—including attestation keys—despite Intel's MDS mitigations. These showed that cache side-channels can compromise not just data confidentiality but also core SGX primitives like remote attestation.

Recent works like ScatterCache [138] propose defenses like randomized indexing to hinder eviction-set construction, but practical challenges and performance impacts leave caches an enduring attack vector. As of 2025, research [139] shows that improvements in finding eviction sets keep cache attacks practical and increasingly precise. These attacks remain a fundamental threat to TEEs, reinforcing the need for defense-in-depth strategies when deploying enclaves in privacy-critical workflows.

Cryptanalytic Side-Channels

Cryptanalytic side-channel attacks exploit vulnerabilities in cryptographic operations inside a TEE, with the goal of recovering secret keys, nonces, or plaintexts. Unlike generic cache attacks, these target properties of the cryptographic algorithm itself, such as constant-time implementations or ciphertext-dependent behaviors.

For example, **CIPHERLEAKS** [113] reveals a ciphertext side-channel in AMD SEV, including SEV-ES and SEV-SNP, where identical plaintexts at fixed physical addresses always produce the same ciphertext. A malicious hypervisor can observe these ciphertext changes in encrypted register save areas (VMSA) during VMEXITs, recovering private keys from constant-time RSA and ECDSA implement-

ations.

ChiperSteal [140] demonstrates a related attack leaking input data of TEE-protected neural networks by observing ciphertext patterns during inference. Similarly, **TraceAttacksSGX** [141] shows that fine-grained control-flow and timing observations can recover RSA keys from a single SGX key-generation trace, despite constant-time implementations.

These attacks underscore that constant-time code alone cannot prevent side-channels when hardware or TEE designs leak information correlated with secret-dependent operations. Since these attacks specifically exploit cryptographic implementations rather than general cache behavior, they merit separate analysis when assessing TEE security for protocols requiring robust key confidentiality.

Power Side-Channels

Power side-channel attacks exploit correlations between a CPU's power consumption and the data or instructions it processes. While traditional attacks relied on physical oscilloscopes, modern CPUs expose interfaces like Intel's Running Average Power Limit (RAPL), effectively turning the processor into its own power meter and enabling software-based power attacks without hardware probes.

PLATYPUS [142] showed that unprivileged processes could measure energy consumption through RAPL, observing data-dependent variations even in SGX enclaves. Combined with precise control of enclave execution using SGX-Step, attackers could statistically recover cryptographic keys, control flow details, and kernel data by correlating power measurements with instruction execution. This revealed that energy consumption varies with the Hamming weight of operands, undermining constant-time implementations.

Collide+Power [143] extended these attacks by exploiting subtle leakage when attacker and victim data are co-located in caches or buffers. By varying attacker-controlled guesses and measuring power or timing variations (e.g., caused by frequency throttling), they performed differential power analysis to recover victim data at rates up to 4.8 bits/hour. Their work established that co-location alone suffices to leak secrets, even without cryptographic vulnerabilities.

Recent surveys [144] highlight that adaptive power management introduces timing variations reflecting subtle power differences, allowing attacks even if RAPL is disabled. Privileged attackers, such as malicious kernels, can exploit these channels to compromise TEEs like SGX or SEV. Power side-channels thus represent a fundamental threat, combining accessibility and precision sufficient to extract secrets from protected environments.

3.6.2 Transient Execution Attacks

Transient execution attacks exploit speculative or out-of-order instructions that execute before permission checks or branch resolutions complete, leaving microarchitectural traces despite being squashed architecturally. This enables attackers

to leak secrets via side channels that measure timing, cache state, or execution resource usage [145, 146].

SMT-based covert channels. Simultaneous Multithreading (SMT) allows multiple[8] logical cores to share the execution resources (e.g. ports and functional units) of a single physical core. This design improves utilization, as many execution units would otherwise remain idle while waiting on the (much) slower memory subsystem. Early research such as Covert Shotgun [147] demonstrated that contention on shared SMT resources can be systematically explored to identify covert communication channels. Building on this, an attacker co-located on the same physical core as a victim can measure port contention or execution latency to infer victim activity. PORTSMASH [148] shows how port contention can reveal fine-grained execution behavior, while SMoTherSpectre [149] demonstrates that speculative execution amplifies SMT-based channels by leaking transient data through shared resources. IChannels [150] further systematizes such attacks, highlighting their effectiveness even against enclaves.

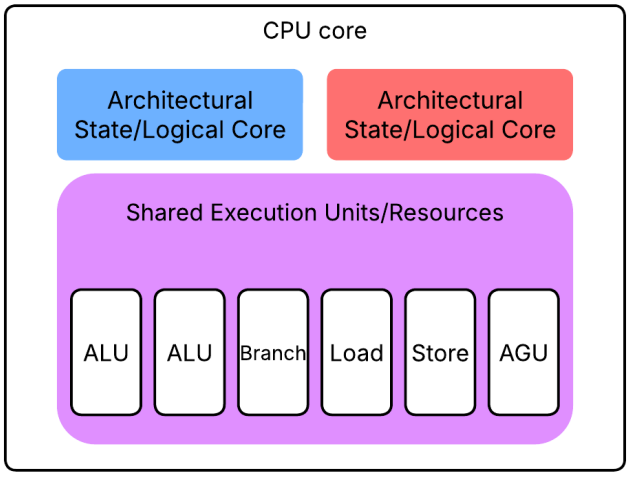


Figure 3.12: Conceptual view of SMT: two logical cores with separate architectural state share execution units, enabling adversaries to observe contention and establish covert channels.

Branch prediction attacks. Branch predictors [151, 152], including the Branch Target Buffer (BTB) and Return Stack Buffer (RSB), speculate on control flow to keep pipelines full. By mistraining predictors, an attacker can cause a victim to speculatively execute unauthorized memory accesses. BranchScope [153] shows directional branch predictors can leak control-flow information. SpectreRSB [154] targets speculative returns by manipulating the RSB, forcing transient execution along attacker-chosen paths. SgxSpectre highlights that these mechanisms can leak secrets even from SGX enclaves [155], while PowSpectre explores attacks against power-efficient cores with shared predictors [156].

[8] Typically two on most modern commodity processors.

Meltdown and Foreshadow. Unlike Spectre's misprediction-driven paths, Meltdown exploits deferred exception handling: the CPU may speculatively fetch data before a permission check completes. This data influences microarchitectural state, e.g., by caching forbidden values. When an exception is raised, architectural state is corrected, but caches still hold evidence of transiently accessed secrets [157]. Meltdown can leak data across kernel-user boundaries, violating hardware-enforced privilege isolation.

Foreshadow (L1 Terminal Fault) extends this by exploiting transient forwarding of invalid page translations. Instead of causing faults on disallowed access, the CPU forwards data from L1 caches transiently, leaking contents of SGX enclaves or virtual machines [158, 159]. Unlike Meltdown's linear user-kernel leak, Foreshadow enables an attacker to target specific physical addresses, breaking confidentiality of enclaves by bypassing page-table protections.

At this point, understanding the nuances and differences between the attacks requires attention. Canella et al. [145] created a taxonomy that categorizes attacks as either Spectre-type (based on misprediction) or Meltdown-type (based on fault handling). Figure 3.13 illustrates this taxonomy.

Microarchitectural Data Sampling (MDS). MDS attacks exploit speculative forwarding of stale data from CPU-internal buffers. RIDL [160] demonstrates that speculative loads can read leftover data from the Line Fill Buffer (LFB), enabling attackers to sample secrets left by other processes or enclaves. ZombieLoad [161] shows how fault-induced transient execution in simultaneous threads causes cross-domain leakage, leaking data across logical cores sharing a physical core. Fallout [162] targets the Store Buffer, leaking metadata about memory accesses and enabling reconstruction of control flows or defeating address randomization. These attacks do not rely on addresses selected by the attacker but opportunistically sample data being processed elsewhere, making them powerful cross-domain leaks.

CacheOut [136] builds on MDS by evicting victim data from L1-D back into the LFB, selectively leaking secrets even with mitigations in place. It shows that buffer-clearing instructions alone may not fully prevent transient sampling. SGAxe [137] further extends this by leaking SGX attestation keys, forging attestation quotes, and completely breaking trust in SGX remote attestation.

Code listing 3.1: ECDSA key recovery example from SGAxe [137]

```
msgHash, r, s = call_enclave()
recovered_entropy = get_leaked_entropy()
z = int(msgHash, 16)
for k in recovered_entropy:
    p = ((s*k - z) * inverse_mod(r, n)) % n
    if attemptSign(msgHash, p, k) == msgHash:
        print("key␣is:␣" + hex(p))
```

Cross-core transient attacks. CrossTalk [163] demonstrates transient leakage across physical cores by abusing shared staging buffers used during off-core data requests. Unlike MDS attacks limited to same-core scenarios, CrossTalk shows that

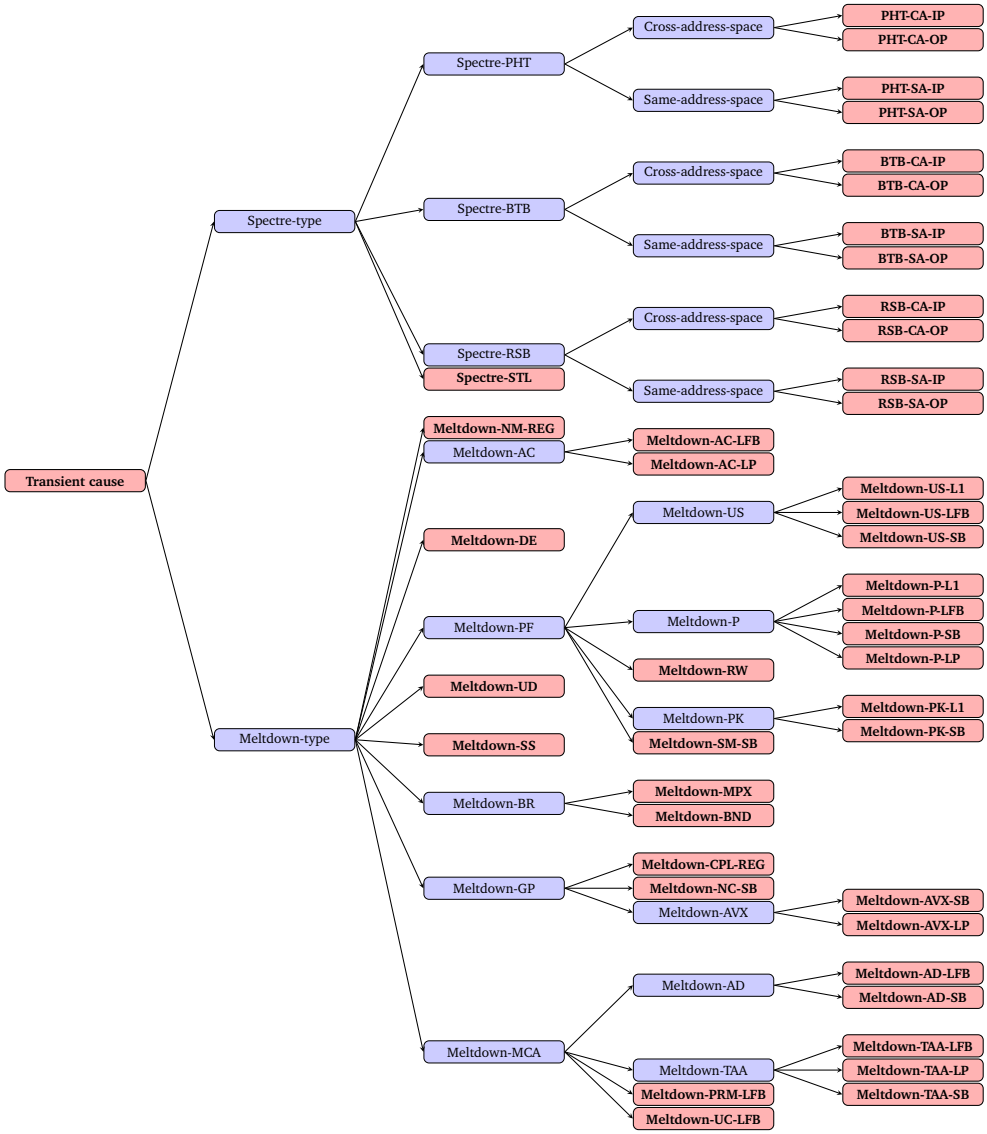


Figure 3.13: Taxonomy of transient execution attacks, as developed by Canella et al. [145]. The taxonomy splits attacks into Spectre-type (based on misprediction) and Meltdown-type (based on faults or assists). An interactive version can be found at `https://transient.fail/`.

these shared buffers allow leakage even when the victim and attacker are scheduled on separate cores. It's exploit against SGX shows full recovery of enclave signing keys from a single observed signature, demonstrating a complete compromise of attestation security.

Emerging threats. Attacks like Load Value Injection [164] (LVI) reverse the data flow of Meltdown: instead of reading unauthorized data transiently, the attacker injects malicious data into victim speculative execution, influencing enclave operations. More recently[9], a novel attack named Branch Predictor Race Conditions (BPRC) was disclosed by researchers at ETH Zurich [165]. It enables unprivileged code to influence prediction for privileged code or enclaves even on CPUs with Spectre v2 mitigations. This is a prime example of the on-going and evolving nature of the field, and future attack vectors cannot be disregarded.

Overall, transient execution attacks demonstrate how speculative optimizations in modern CPUs fundamentally conflict with strong security guarantees, including those promised by TEEs like SGX. These vulnerabilities persist despite ongoing microcode and OS-level mitigations, emphasizing the need for architectural redesigns or complementary defenses when deploying trusted environments on commodity hardware.

3.6.3 Software-Based Fault Attacks

This section focuses on fault attacks that can be triggered through software, either by exploiting exposed interfaces like DVFS or by manipulating the memory subsystem from unprivileged code. These attacks do not require invasive physical access but instead abuse system features to induce faults in protected computations. This category includes undervolting attacks on computation units, clock manipulation, and disturbance-based memory corruption. Although the mechanisms vary, the common characteristic is that the fault is initiated from software running on the target platform.

Voltage-based Fault Injection. Dynamic Voltage and Frequency Scaling (DVFS) allows CPUs to adjust power consumption for performance and efficiency. Voltage fault attacks exploit the fact that undervolting can lead to timing violations in computation units. **Plundervolt** [166] demonstrated that, on Intel SGX, maliciously reducing core voltage can flip bits in secure enclave computations, allowing attackers to recover AES keys through induced faults. **V0LTpwn** [167] simialrily demonstrated undervolting as a technique to extract cryptographic keys from SGX enclave. **VoltPillager** [168] later showed a physical[10] version by controlling voltage externally over the Serial Voltage Identification (SVID) bus, which bypassed firmware-based protections and allowed extracting AES keys from SGX

[9] This attack was published during the writing of this thesis.

[10] While VoltPillager requires physical access to the motherboard, it is included here for completeness due to its conceptual similarity to software-based undervolting attacks.

enclaves. On AMD SEV, **One Glitch to Rule Them All** [168] exploited voltage manipulation of AMD-SP to forge attestation reports, completely compromising SEV's RoT and allowing attackers to fake secure measurements. Likewise, **VoltJockey** [169] showed a practical undervolting attack against both TrustZone and SGX, enabling controlled bit flips in cryptographic routines to compromise keys.

These attacks exploit that CPUs only check voltage integrity at the firmware/-BIOS level or not at all during runtime; therefore, even TEEs assuming a strong hardware RoT are vulnerable if an attacker can manipulate supply voltage or exploit firmware weaknesses. Mitigations like disabling dynamic voltage scaling or enforcing runtime voltage integrity via hardware security modules have been proposed [121].

Clock-based Fault Injection. Clock fault injection leverages the CPU's frequency scaling mechanisms. By manipulating clock speeds outside expected operating parameters, attackers can cause instruction skipping or timing-related faults. **CLKSCREW** [170] demonstrated that controlling Dynamic Voltage and Frequency Scaling on ARM-based processors can break TrustZone by forcing erroneous computations or bypassing checks, enabling root-level exploits and compromising trusted applications. This attack highlights the risk of relying on firmware-enforced clock control in TEEs.

Challenges and Mitigations. Voltage and clock fault-injection attacks reveal a key limitation in many TEEs' assumptions: they implicitly trust the processor's voltage and frequency regulators. As surveyed in [70, 121], TEEs require mechanisms like constant voltage monitoring, hardware-based overclock/undervoltage detection, and fail-safe mechanisms that can halt execution upon anomaly detection. Unfortunately, these mitigations either introduce substantial cost or performance penalties or require redesigns of existing CPU architectures.

DRAM Disturbance: Rowhammer

Rowhammer [171] fundamentally changed the way researchers think about memory isolation. It demonstrated that software alone can induce hardware-level bit flips in DRAM, thereby violating one of the most basic assumptions in system security: that physical memory is passive and cannot be corrupted by unprivileged code. At a high level, repeatedly accessing (or "hammering") rows adjacent to a "victim" row can cause charge leakage and, eventually, bit flips in that victim row. This effect arises due to the high density and low noise margins of modern DRAM.

The emergence of vulnerable DIMMs can be traced back to around 2012–2013 [171], coinciding with the transition to smaller DRAM process nodes. Since then, a wide body of research has explored not just how to trigger bit flips reliably, but also how to exploit them across increasingly abstracted environments—from native binaries to JavaScript sandboxes [172]. DRAM is organized into channels, ranks, banks, and rows, with each row typically being 8 KB in size. When a row is

accessed, it is loaded into the row buffer—a kind of internal cache. If two different rows in the same bank are accessed in rapid succession, the repeated activation of the row buffer can induce enough disturbance to flip bits in a nearby row.

Rowhammer attacks have evolved well beyond flipping a single bit in user-space memory. Advanced exploitation techniques can flip bits in page tables, gain kernel privileges, or even manipulate enclave memory. For example, SGX-Bomb [173] showed how Rowhammer can corrupt the EPC metadata to cause enclaves to crash or behave incorrectly. Nethammer [174] demonstrated that Rowhammer can be triggered remotely through network traffic alone, making it a potential vector for fault injection even in cloud-hosted enclaves. In some cases, flipping bits in enclave-associated memory can bring down the system entirely or cause persistent data corruption.

Despite mitigation attempts such as ECC, Target Row Refresh, and software-based detection, Rowhammer continues to be a concern. As some reachers argue [172], this is not just a hardware bug—it is a fundamental shift in how we must reason about trust boundaries in hardware design. Rowhammer's longevity as a research topic speaks to its continued relevance, both as a practical threat and as a mechanism for inducing controlled faults, including in TEEs [175].

3.6.4 Architectural Design Flaws

Architectural design flaws in TEEs arise when the system's trust model or runtime does not adequately account for the capabilities of an untrusted OS or hypervisor. A recurring theme is the insecure handling of external events—whether interrupts, exceptions, or signals—delivered by a potentially malicious management layer. Because enclaves and confidential VMs rely on the OS or hypervisor to deliver these events, they implicitly trust the timing, frequency, and correctness of notifications.

A key example of this class is the family of attacks exploiting *malicious notification injection*. **SmashEx** [176] exposed how Intel SGX's asynchronous exception handling can break enclave state consistency. In SGX, when an enclave receives an interrupt, it performs an AEX and resumes execution via `ERESUME`. If the OS or hypervisor carefully times exceptions, it can cause an enclave to re-enter at unintended points, reusing stale register or memory state. Attackers can combine these forced re-entries with code-reuse (ROP) or speculative gadgets to extract secrets without needing memory safety vulnerabilities. **AsyncShock** [177] extended this by identifying bugs in synchronization between enclave exits and resumes, showing that malicious AEX delivery can cause invariant violations in SGX runtimes, potentially leading to memory corruption or unintended enclave control flow.

Virtualization-based TEEs face similar pitfalls. **HECKLER** [178] showed that even though a hypervisor cannot decrypt a Confidential VM's memory, it can manipulate virtual interrupts to control the VM's execution path. By monitoring page-level execution (using page faults as a primitive) and strategically injecting interrupts, the attacker can induce the guest to misinterpret control flow, bypass au-

thentication steps, or execute attacker-chosen instruction sequences. This enables high-impact attacks like bypassing OpenSSH or sudo authentication inside Confidential VMs. **WeSee** [117] demonstrated a parallel in AMD SEV-SNP, targeting the #VC exception mechanism designed for secure communication between VM and hypervisor. By crafting #VC messages with malicious exit reasons, a hypervisor can trick the guest OS into revealing secrets or altering critical kernel structures, exploiting the lack of strict semantic validation in #VC handlers. **Sigy** [179] revealed that similar notification abuse is possible even in SGX, where the OS can legally deliver POSIX signals (e.g., `SIGFPE`) to enclave threads. By injecting signals at precise execution points, attackers can force enclaves to handle them under attacker-chosen conditions, corrupting enclave state or causing the enclave to execute code paths exposing secrets.

Another group of attacks arises from incorrect assumptions about runtime and ABI setup in enclave or TEE environments, leading to vulnerabilities even without classic memory bugs. **COPYCAT** [180] highlighted that SGX's page-fault-based controlled-channel attacks can be amplified when the OS combines them with precise single-stepping via frequent interrupts. By counting the number of instructions between page accesses, an attacker can reconstruct enclave control flow at instruction granularity, even when cryptographic code uses constant-time techniques—effectively creating a single-trace side channel powerful enough to extract keys.

Faulty Point Unit (FPU) [181, 182] attacks revealed a subtle but dangerous class of vulnerabilities in SGX runtimes and other TEEs. Many runtime libraries fail to properly initialize x87 FPU, SSE, or AVX control registers upon enclave entry, leaving them under OS control. A malicious OS can poison these registers before entering the enclave, modifying floating-point behavior by changing precision, rounding modes, or unmasking exceptions. This allows attackers to induce computation errors or convert floating-point exceptions into precise, attacker-controlled fault channels—subverting enclave confidentiality and integrity without needing to break memory safety or cryptographic primitives.

A final example of an architectural design vulnerability is *Æ*PIC Leak [183], which shows how hardware register design can directly compromise enclave confidentiality. On Intel CPUs affected by *Æ*PIC Leak, reserved offsets in the Local APIC MMIO range return stale data from the CPU's internal buffers instead of triggering faults or returning zeroed values. A privileged attacker running with OS-level control can read these reserved registers to retrieve architecturally exposed data, including sensitive information from SGX enclaves such as AES, RSA, or even attestation keys. This attack is reminiscent of some of the examples discussed in Section 3.6.2, does not rely on speculative execution or timing side channels, but instead uses architecturally defined register reads to access stale microarchitectural state. *Æ*PIC Leak demonstrates that improper initialization or handling of architectural registers can allow direct, reliable extraction of enclave secrets, undermining TEE security even when external event handling and runtime protections are implemented correctly.

Together, these attacks show that architectural design flaws often stem not only from misplaced trust in the OS or hypervisor, which can control key events or architectural state transitions, but also from hardware-level issues such as incorrect register initialization. Even when enclaves or confidential VMs maintain strict memory isolation, assumptions about secure external event delivery, proper runtime re-entry handling, or correct architectural register behavior can undermine security guarantees. Careful design and verification of both TEE runtimes and hardware implementations is necessary to ensure isolation and confidentiality.

3.6.5 Physical Attacks

Physical attacks on TEEs exploit the opportunity for an adversary with hands-on access to the hardware, such as a rogue or coerced data center employee, to bypass software-based protections. These attacks differ from software or logical exploits in that they require direct manipulation or observation of hardware components. It is also worth noting some attacks, for instance VoltPillager [168] described in Section 3.6.3, sit in a grey area between purely software-based fault injection and fully physical attacks.

The Off-Chip Attack via the memory bus, known as MEMBUSTER [184], demonstrates that by attaching a physical interposer device to a server's DRAM DIMM slot, an attacker can passively observe signals on the memory bus. Even when memory encryption is in place, metadata like memory addresses often remain unencrypted for performance reasons. By capturing these address lines, the attacker can reconstruct precise memory access patterns of an enclave or confidential VM. This level of visibility reveals secret-dependent data flows or control decisions, leaking information without needing to tamper with software or inject faults.

In BadRAM [185], researchers show that by repeatedly activating specific DRAM rows, an attacker can induce bit flips in adjacent rows, a phenomenon called rowhammer. By carefully selecting target rows and crafting memory layouts, they can cause bit flips in enclave page tables or critical data structures. This corrupts pointers inside the enclave, which may result in arbitrary memory reads or writes within the protected address space, ultimately allowing attackers to leak or modify enclave secrets. Although rowhammer can be triggered by software, reliably achieving target bit flips often benefits from physical techniques such as manipulating temperature or clocking conditions to increase DRAM susceptibility.

These attacks highlight that physical access to servers hosting TEEs opens a powerful attack vector, allowing adversaries to bypass logical protections by exploiting the electrical or structural properties of hardware itself. Even in cloud environments where TEEs are used to protect data from malicious system software, the underlying hardware must be physically secured to maintain meaningful confidentiality guarantees.

Chapter 4

Methodology

In this chapter, we detail the systematic methodological approach undertaken to address the research objectives. We outline the process for conducting the comprehensive literature review that forms the foundational knowledge. Furthermore, we precisely define the scope of the research, elaborate on the methodology employed for designing the architecture and protocol, and critically assess the inherent limitations of the chosen approach.

4.1 Literature Review

The literature review established a foundation for the thesis by synthesizing current knowledge on TEEs, PETs, and secure system design. We adopted a systematic approach inspired by recognized best practices in computing research [186, 187], starting with a clearly defined scope focused on the feasibility of using TEEs for privacy-preserving record linkage in cloud environments.

As a starting point, I leveraged foundational knowledge gained through coursework in System Security and Hardware Security during an exchange at ETH Zurich. These materials provided both theoretical grounding and practical entry points into ongoing research discussions, which helped guide and focus the review process.

Academic databases including IEEE Xplore, ACM Digital Library, ScienceDirect, and arXiv were searched to capture peer-reviewed publications and high-quality preprints. Repositories such as the UN PET Lab and the Confidential Computing Consortium were consulted for practitioner perspectives and recent white-papers. Keywords used in various combinations included: “trusted execution environment”, “TEE”, “Intel SGX”, “AMD SEV”, “confidential computing”, “privacy-enhancing technologies”, and “record linkage”. Priority was given to papers from recent top-tier conferences such as USENIX Security, IEEE S&P, and NDSS, given the fast-paced development of TEEs.

Titles and abstracts were screened first to exclude irrelevant works. Full texts were then reviewed to assess relevance to the research questions, with a focus

on papers addressing practical deployment, privacy-preserving system architectures, and protocols combining TEEs with statistical analysis. Grey literature was included selectively when it offered insights not yet present in formal publications, such as vendor developer guides or consortium specifications. To mitigate potential vendor bias, claims in hardware documentation were cross-checked with independent academic analyses or security evaluations whenever possible.

The synthesis process identified recurring themes such as TEE threat models, attestation workflows, side-channel vulnerabilities, and integration strategies with privacy-preserving protocols. Foundational works, such as Schneider et al.'s Systemization of Knowledge on TEEs [16], helped clarify key architectural properties. Information from multiple sources was correlated to build a coherent understanding of platform features, limitations, and security considerations.

4.2 Scope

We limit the detailed investigation to commercially available x86-based TEEs, specifically Intel SGX and AMD SEV, as these are supported by major cloud providers and are the most realistic options deployments today. Although other TEEs[1] are briefly described, they were not analyzed in depth. Including them would have significantly expanded the scope without contributing proportionally to our focus on practical, near-term deployment scenarios.

On the statistical side, the analysis is centered on record linkage, a task where privacy concerns are well-established. Limiting the investigation to this domain allows for a focused, technically detailed discussion of how TEEs can enable input privacy for a concrete, high-risk operation in official statistics, without diluting attention across unrelated statistical techniques.

These scope decisions were made to balance relevance, technical depth, and feasibility, ensuring the thesis could offer actionable insights into real-world use cases while staying grounded in current commercial technology.

4.3 Protocol Design Methodology

For the protocol design methodology we combined iterative reasoning with practical guidance from real-world systems and vendor documentation. While formal threat modeling frameworks like STRIDE or Dolev-Yao were not applied exhaustively, we conceptually considered common attack vectors identified in TEE research such as spoofing of enclaves, data tampering in transit, and side-channel risks, and sought to minimize the TCB by simplifying enclave responsibilities.

Our design approach was grounded in the UK National Cyber Security Centre's (NCSC) protocol design principles [188], which emphasize focusing on the use case, keeping the design as simple as possible, and accounting for the broader ecosystem. Starting from a high-level outline, we iteratively refined a protocol

[1] Open-source academic TEEs are mentioned in).

where multiple independent data owners contribute inputs to a TEE-hosted computation that aggregates data securely without exposing individual records.

Throughout the design process, we consulted practical examples from enclave-based systems such as VC3 [189]. These provided valuable lessons on secure provisioning, remote attestation, and efficient data handling inside enclaves. Vendor developer guides, such as the Intel SGX Developer Reference, were used to align protocol steps with platform-specific constraints and attestation flows. These practical references helped ensure the protocol design remained grounded in real-world capabilities.

4.4 Limitations of Methodology

This methodology is inherently theoretical. No prototype implementation was created or empirically validated. As such, performance characteristics, usability challenges, and integration complexities remain untested, leaving potential gaps in practical feasibility. While the design decisions were informed by vendor documentation, academic literature, and existing implementations, they have not been subjected to formal security proofs or adversarial testing, leaving room for undiscovered vulnerabilities.

Additionally, TEEs continue to evolve rapidly, and incomplete or proprietary documentation particularly on aspects like microcode-level protections or attestation ecosystem details, introduces uncertainty. Assumptions made in the protocol design may be invalidated by future hardware updates or newly discovered vulnerabilities. Therefore a (brief) reassessment should be conducted before any real-world deployment.

Finally, organizational and legal considerations, such as establishing trust agreements among NSOs or ensuring compliance with data protection regulations, were outside the scope of this methodology. These socio-technical factors can be decisive in practice, but were not addressed here.

Overall, while the methodology combines systematic literature synthesis with iterative design grounded in practical realities, it should be understood as a conceptual foundation for future work, requiring empirical validation and potential adjustments before operational use.

Chapter 5

Feasibility Analysis

In this chapter, we evaluate whether it is technically feasible to build trusted systems for multi-party statistical computation using TEEs today, with a focus on Intel SGX. We examine current confidential computing offerings from major cloud providers, explore real-world deployments, and analyze enclave development complexities. Our analysis highlights opportunities and challenges, providing essential insights to guide the system design in later chapters. While grounded in technical realities, we also consider organizational perspectives relevant to NSOs.

5.1 Confidential Computing Cloud Solution Providers

Leading Cloud Service Providers (CSPs) have integrated confidential computing capabilities into their platforms using TEEs such as Intel SGX, AMD SEV, or Intel TDX. Although all major CSPs claim to support confidential workloads, the depth of their services varies. This section compares Microsoft Azure, Google Cloud Platform (GCP), and Amazon Web Services (AWS) as of mid-2025, emphasizing practical TEE support, developer tooling, and attestation services. These details reflect offerings as of mid-2025 and may change as providers update their services.

Microsoft Azure

Microsoft has been a key proponent of confidential computing, coining the term and helping found[1] the Confidential Computing Consortium (CCC). It maintains the Open Enclave SDK [2], an open-source abstraction layer for developing applications across multiple TEEs including Intel SGX and ARM TrustZone, facilitating portable enclave code [190]. Azure also provides Azure Attestation, a managed service for attesting SGX enclaves, TPMs, and virtualized environments, allowing developers to define attestation policies via claims and evidence [191].

[1] Intel, Google and Microsoft are among the founding premier members of the Confidential Computing Consortium: `https://confidentialcomputing.io/about/members/`.

[2] `https://openenclave.io/sdk`

Azure offers DCsv2- and DCsv3-series VMs, which support SGXv1 and SGXv2 respectively. The DCsv3-series supports enclaves up to 256 GB, addressing previous scalability limitations. Although these VMs are not currently available in Norwegian data centers, they are offered in nearby regions like North and West Europe. Azure also provides preconfigured confidential VM images, as well as managed services such as the Confidential Consortium Framework [192], supporting decentralized applications with verifiable execution inside enclaves.

Overall, Azure provides the most comprehensive confidential computing environment among major CSPs, with mature tooling and flexible enclave support suitable for general-purpose secure computation.

Google Cloud Platform

GCP, also a founding premier member of CCC, partners with Intel and AMD to offer confidential VMs built on TDX and SEV-SNP [193, 194]. These VMs allow users to secure existing applications without modification, but they do not expose developer-level enclave APIs comparable to SGX. GCP's SGX SDK, Asylo[3], appears unmaintained, with its last commit over three years ago at the time of writing. Google's recent resources, such as talks at industry conferences and blog posts, focus exclusively on confidential VMs[4], without any mention of SGX.

Although GCP's Confidential VMs are broadly available and easy to deploy, the lack of maintained SDKs or enclave-level APIs could render them less suitable for use cases requiring secure multiparty computation or detailed trust models.

Amazon Web Services

AWS does not offer hardware-based TEEs such as Intel SGX, AMD SEV-SNP, or Intel TDX [195, 196]. Instead, it provides Nitro Enclaves, which use AWS Nitro's dedicated hardware and lightweight hypervisor to enforce isolation. Nitro Enclaves do not include microarchitectural protections or CPU-enforced memory encryption; rather, they rely on virtualization boundaries enforced by the Nitro Hypervisor. Attestation is supported through the Nitro Hypervisor, which provides signed enclave measurements for verification. AWS also offers SDKs[5] in C and Rust, enabling vsock communication and attestation integration with AWS services.

Nitro Enclaves are best suited to narrowly scoped tasks like secure key management or payment processing, but lack the fine-grained enclave APIs and microarchitectural protections required for general-purpose secure computation or privacy-preserving analytics.

[3] `https://github.com/google/asylo`

[4] See, e.g., "How Google and Intel make Confidential Computing more secure" `https://cloud.google.com/blog/products/identity-security/rsa-google-intel-confidential-computing-more-secure`

[5] `https://github.com/aws/aws-nitro-enclaves-sdk-c`

5.2 Existing Use-Cases and Implementations

TEEs have evolved from academic prototypes into production-ready technologies, with real-world deployments demonstrating their feasibility for privacy-preserving computation in sensitive domains like health, finance, and official statistics. This section highlights representative examples (not an exhaustive list) illustrating how TEEs are already securing collaborative analytics in practice.

Initial feasibility was established by academic systems such as VC3 [189], which demonstrated secure MapReduce jobs inside SGX enclaves, and Panoply [197], which reduced the TCB required for running unmodified Linux binaries. While these prototypes laid important groundwork, the examples below focus on real-world deployments processing sensitive data outside lab environments.

Intel lists numerous partners[6] using SGX for confidential data collaboration. For example, **Aggregion** built a platform enabling retailers like Magnit to jointly analyze customer data while keeping each party's input isolated inside SGX enclaves on Azure [198]. Microsoft highlighted this deployment as a practical case of privacy-preserving data sharing across organizations.

Commercial platforms such as **Decentriq** and **BeeKeeperAI** (both Intel partners) extend these capabilities. Decentriq's "data clean rooms" leverage enclaves for joint analytics in healthcare and finance, including biomedical AI projects under the EU-funded SEARCH initiative [199], though detailed technical documentation is not publicly available due to non-disclosure agreement restrictions. BeeKeeperAI, developed at UCSF's Center for Digital Health Innovation with Microsoft and Intel, allows hospitals and algorithm developers to train AI models on sensitive patient data in SGX enclaves without revealing raw records or proprietary algorithms [200], implementing a "Zero Trust" approach to meet healthcare privacy requirements.

In the public sector, governments and international agencies have piloted TEE-based statistical pipelines to address privacy and legal constraints in cross-organizational data use. Several case studies curated by the UN Statistics Division [201] demonstrate the use of SGX-backed infrastructure in operational settings. For instance, the Ministry of Tourism in Indonesia securely combined telecom data from two mobile network operators to produce roaming statistics without exposing raw subscriber records. In Europe, Eurostat employed TEEs to process longitudinal mobile network operator data while maintaining legal confidentiality, and in another case, the UN PET Lab facilitated cross-border analysis of international trade data, contributed by several national statistics offices [201]. These pilots collectively illustrate the feasibility of enclave-based computation in statistical workflows that span organizational and jurisdictional boundaries.

Additionally, the **Royal Society**'s 2023 policy report [202] highlighted efforts by the UK Office for National Statistics to explore TEEs for secure collaboration across government agencies. Together, these examples show TEEs solving real-

[6] `https://www.intel.com/content/www/us/en/architecture-and-technology/sgx-product-offerings.html`

world privacy and trust challenges in statistical analysis and secure data sharing. While proprietary implementations often limit external scrutiny, their existence confirms that enclave-backed computation is technically and commercially feasible today for trusted statistical systems.

5.3 Development and Implementation

In this section we analyze the practical development options for Intel SGX on Linux, highlighting the challenges developers face when building secure enclave applications today. It draws on the May 2025 versions of the *SGX Developer Guide* [203], *SGX Developer Reference* [204], and *Volume 3D: System Programming Guide* of the Intel SDM [205].

We compares two main development approaches: partitioning-based enclave design, where developers manually separate trusted and untrusted code, and library OS solutions that enable running legacy applications with minimal modification. These models shape the trade-offs between TCB size, attack surface, and engineering effort. Following that, we describe key aspects of the development process and

5.3.1 Development Models

Intel SGX enclaves offer strong hardware-based isolation, but leveraging these protections requires significant architectural decisions. Two primary approaches exist: partitioning-based development and Library OS (LibOS) solutions. Table 5.1 compares the main characteristics of these models.

Table 5.1: Comparison of partitioning and library OS development models

Characteristic	**Partitioning Model**	**Library OS Model**
Effort	High manual engineering effort; source access needed	Lower effort; legacy binaries often reused
TCB Size	Small (developer-controlled)	Large (LibOS + runtime)
Security	Smaller TCB but larger host interface	Reduced host interface but larger enclave TCB
Tools	Intel SGX SDK, OpenEnclave, Rust SGX SDK	Gramine-SGX, Occlum, SCONE
Typical Use	Security-critical modules	Unmodified legacy applications

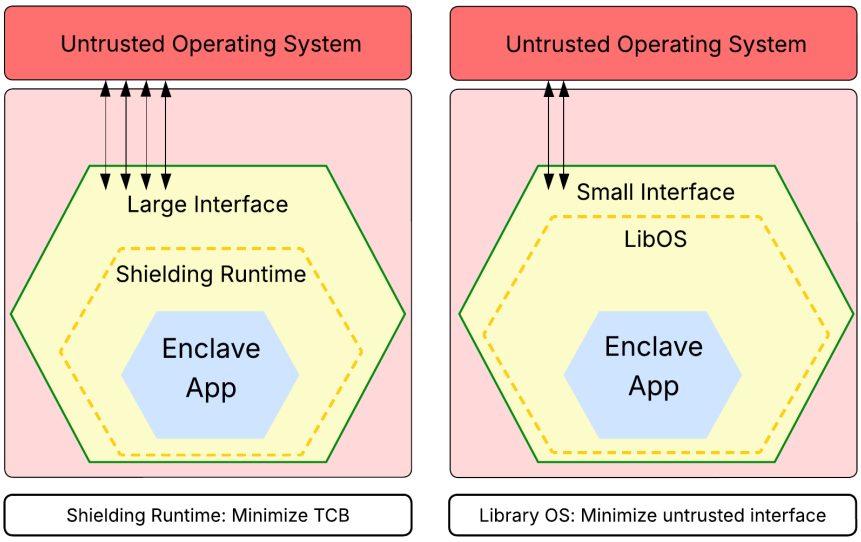


Figure 5.1: SGX development spectrum. Left: minimal shielding runtimes with small TCBs but large untrusted interfaces. Right: LibOS-based runtimes embed OS features inside the enclave, reducing external interface size at the cost of a larger TCB. Further details appear in Section 7.2.1.

Aside: Runtime Environment

A runtime provides essential support for executing a compiled program. For instance, in a typical C program on Linux, execution begins at the linker-defined entry point `__start`. This calls `__libc_start_main`, responsible for initializing arguments, environment, and system libraries. Only then does control pass to the programmer-defined `main` function.

Shielding runtimes and the partitioning-to-LibOS spectrum. As shown in Figure 5.1, all SGX applications rely on shielding runtimes to manage secure transitions, sanitize inputs, and bridge API/ABI differences between untrusted hosts and trusted enclaves. Minimal runtimes (e.g., Intel SGX SDK, Open Enclave SDK) require explicit ECALL/OCALL definitions, resulting in a larger attack surface. LibOS solutions (e.g., Gramine [206], SCONE [83], Haven [75]) embed richer OS abstractions inside the enclave, which reduces the untrusted interface but significantly increases the TCB. These trade-offs fundamentally shape complexity, security, and feasibility.

Approach 1: Partitioning-based development. Manual partitioning remains the recommended SGX paradigm [203], isolating only security-critical code in the enclave for a minimal TCB. Developers define ECALLs, OCALLs, and manage enclave memory and thread structures. Shielding runtimes inside the enclave

handle marshaling and ABI boundaries, but developers must still identify and separate sensitive logic—a process requiring deep understanding of the application.

Tools like the Intel SGX SDK and OpenEnclave SDK support this model by generating glue code through Edger8r, which parses Enclave Definition Language (EDL) files. However, partitioning requires substantial rewriting or refactoring. Compiler frameworks enable semi-automatic partitioning using annotations and static analysis [207], though these tools depend on correct identification of sensitive components and may not capture complex runtime behaviors. Other toolkits, like OpenEnclave[7] and the Rust SGX SDK [208], provide safer abstractions for partitioned applications but still require manual intervention for security-critical parts. Automatic annotation-driven efforts [207] show promise but remain research prototypes. Overall, partitioning best suits cases where only a small portion of the logic demands confidentiality [209].

Approach 2: Legacy software via LibOS. LibOS solutions like Gramine-SGX [206], Occlum [210], SCONE [211], and Haven [75] enable unmodified binaries to run inside SGX enclaves by replicating OS functionality. These frameworks expose familiar APIs (e.g., POSIX, Win32), treating enclaves like lightweight virtual machines.

LibOS projects differ in balancing TCB size against untrusted interface complexity [212]. For example, SCONE [211] and Panoply [197] minimize TCB by passing most syscalls to the untrusted OS with sanity checks, increasing host interface complexity. In contrast, Haven and Graphene embed more abstractions inside the enclave to reduce host interactions, which lowers attack surface at the cost of a larger TCB. This trade-off is visualized in Figure 5.1.

Despite enabling legacy workloads, LibOSes introduce challenges: compatibility with dynamic linking, multithreading, and risk of side-channel leakage through I/O patterns. Secure attestation is also critical, as simulating enclave interfaces could mislead data providers. Intel lists Gramine and Occlum among supported environments[8], but SGX-based LibOSes still require enclave-aware adaptations, unlike AMD SEV, which runs full VMs without modification [209].

5.3.2 Complexity and Developer Burden

Developing secure applications with SGX remains complex even for experienced engineers. Official documentation includes the 52-page *SGX Developer Guide* [203], the 484-page *SGX Developer Reference* [204], and the 286-page *Volume 3D: System Programming Guide, Part 4* [205], which underscores the sophistication required to implement enclaves correctly. Developers must manage enclave layout, metadata, sealing policies, debug restrictions, and resilience to hardware events

[7] https://github.com/openenclave/openenclave

[8] https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/get-started.html

such as power loss. Binaries must be statically linked to produce signed enclave images, as dynamic linking typically prevents proper measurement and signing.

The SGX programming model challenges conventional software assumptions: enclaves cannot trust inputs, cannot directly invoke system calls, and must handle interrupts and rollbacks gracefully. In SGX1, enclave memory regions must be fully defined at creation with `ECREATE`, `EADD`, and `EEXTEND`, with the enclave measurement finalized via `EINIT` [205]. SGX2 improves flexibility by adding instructions like `EAUG` and `EMODT` for dynamic memory expansion, but developers still bear full responsibility for validating buffers, managing alignment, and securely marshaling data across ECALL and OCALL interfaces [203].

Interrupts during enclave execution trigger an AEX, which saves the CPU state in the SSA associated with the TCS and returns control to the untrusted host. Execution resumes with `ERESUME`, which restores enclave state from the SSA [205]. Although abstracted by the SDK, developers must account for these transitions to ensure robust recovery and to prevent sensitive state from leaking.

Instruction Constraints. Enclave code must avoid using certain hardware instructions that are disallowed inside SGX to maintain strict user-mode isolation. These illegal instructions, summarized in Table 5.2, include privileged operations, VM-exiting instructions, and direct I/O. Encountering any of these causes a `#UD` fault, and porting existing code often requires auditing or rewriting routines that rely on them [205, §37.6].

Table 5.2: Illegal Instructions Inside an SGX Enclave

Category	Instructions
VMEXIT-sensitive	`CPUID`, `GETSEC`, `RDPMC`, `SGDT`, `SIDT`, `SLDT`, `STR`, `VMCALL`, `VMFUNC`
I/O operations	`IN`, `INS(B/W/D)`, `OUT`, `OUTS(B/W/D)`
Privilege-level changes	`Far call/jump/ret`, `INT`, `IRET`, `SYSCALL`, `SYSENTER`, segment register changes: `LDS`, `MOV` to DS/ES/SS/FS/GS, `POP` DS/ES/SS/FS/GS

Source: [205, §37.6]

Taken together, these constraints make it clear that implementing secure and efficient SGX applications demands significant engineering effort, specialized knowledge, and ongoing maintenance as hardware and SDKs evolve.

5.3.3 Adaptation Overhead

Integrating existing software into SGX enclaves is rarely straightforward. Applications must be modular enough to separate security-critical logic from components relying on frequent I/O or system calls, which cannot run directly inside the

enclave. Programs that depend on shared memory, dynamic code generation, or extensive OS services typically require extensive redesign to comply with SGX's strict isolation constraints.

Redirecting I/O operations through OCALLs demands enclave-safe wrappers that sanitize data exchanged with the untrusted environment, adding both complexity and performance overhead. Language runtimes such as Python and Go further complicate integration by introducing features like garbage collection and just-in-time compilation, which require enclave-specific adaptations to manage memory safety and control execution flow [208]. Even in traditionally low-level languages like C or Rust, developers often need to construct custom bindings or carefully audit third-party libraries to ensure safe operation inside enclaves.

In SGX1, buffer marshaling must be fully defined at enclave creation using EDL files and enclave-aware allocators with strict alignment guarantees. Although SGX2 introduced instructions like `EAUG` and `EMODT` to enable dynamic memory expansion and permission changes [205, §39.2], these improvements still place responsibility on developers to manage buffer boundaries, alignment, and the exposure of enclave memory to untrusted code.

Taken together, these requirements mean adapting legacy or complex applications for SGX is rarely trivial; rather, it demands careful architectural planning, thorough knowledge of the enclave programming model, and ongoing maintenance as platform constraints evolve. As discussed in the following sections, these burdens also contribute to vendor dependence, since many of the required components and certifications are controlled by Intel.

5.3.4 Enclave Distribution and Versioning

Maintaining secure deployments over time requires mechanisms to ensure continuity across enclave updates. SGX achieves this through its identity system based on the `MRSIGNER` value: enclaves signed with the same key derive consistent sealing and attestation keys, enabling new versions of an enclave binary to access data sealed by earlier versions. This design allows updates and security patches without disrupting access to previously provisioned secrets.

Combined with the Security Version Number (SVN), this mechanism provides basic rollback prevention: newer enclaves (with identical `MRSIGNER` and equal or higher SVN) can read sealed data from older versions, while older enclaves cannot decrypt data created by newer, higher-SVN enclaves [205, §40.8.1]. Together, these features enable persistent secure storage, controlled updates, and flexible deployment strategies, which are essential for maintaining long-term trusted systems.

5.3.5 Vendor Dependence and Lock-In

Despite Intel's decision to open-source significant portions of the SGX SDK, critical components of the SGX ecosystem remain proprietary. Architectural enclaves responsible for provisioning and quote generation, for example, are only distributed

as closed binaries to platform vendors or trusted partners. Accessing advanced features such as production-grade attestation or long-term key provisioning requires integration with Intel's backend infrastructure and strict compliance with platform policies.

These opaque dependencies hinder independent verification of SGX's security guarantees, limiting transparency vital for government agencies or regulated industries that rely on auditable trust anchors. Moreover, SGX platform support is closely tied to specific CPU generations and firmware provisioning certificates. If Intel discontinues support for a processor line or revokes certificates, existing enclave deployments could become inoperable or require costly hardware replacements — underscoring the risk of centralized vendor control.

Practically, SGX's architecture enforces vendor lock-in: applications developed using SGX's fine-grained partitioning model are incompatible with alternative TEEs like AMD SEV-SNP or Arm TrustZone, which differ fundamentally in isolation mechanisms and memory management [209]. While cross-TEE abstraction frameworks such as OpenEnclave aim to ease portability, practical compatibility remains limited and highly dependent on application-specific assumptions [81].

5.3.6 Estimated Performance Overhead

We note a few performance-related considerations here, as it may affect how enclaves are developed. The mechanisms that provide isolation also introduce measurable performance overheads. For SGX, the limited EPC is a major constraint. Studies show that once workloads exceed EPC capacity, performance can degrade by up to two orders of magnitude due to the cost of encrypted paging [213, 214]. Although SGX2 supports larger EPC sizes (up to 512 GB per socket), real-world deployments often configure far less (e.g., 64–128 GB), keeping paging overhead relevant [215].

Frequent enclave transitions (ECalls/OCalls) introduce latency from context switches, TLB flushes, and cache invalidations, with each transition costing around 7,000–8,000 CPU cycles [214]. Protected I/O APIs like `sgx_fwrite` also incur fixed overheads, making batching essential for performance.

Alternative memory models such as Elasticlave [216] propose more efficient cross-enclave sharing to reduce copy overheads, offering throughput gains in data-intensive workloads.

[9] SGX2 on newer Xeon platforms lifts some EPC constraints by using AES-XTS encryption without Merkle-tree integrity checks, trading security for larger enclave memory capacity (128GB to 1TB depending on the processor model [84]); see Section 7.1.3 for security implications.

Table 5.3: Summary of SGX Development Challenges

Challenge	**Implication**
Limited EPC size[9]	Causes performance bottlenecks due to frequent page faults and memory evictions
Manual partitioning	Requires significant engineering effort and access to source code
Insecure interface boundaries	Makes robust input validation and careful interface design essential for security
Tooling complexity	Involves numerous configuration steps, ABI considerations, and signing procedures
Closed-source components	Hinders auditability and complicates deployment in sensitive environments
Vendor dependence	Creates incompatibility with other TEEs and risks future hardware or certificate support changes
I/O restrictions	Forces indirect communication via OCALLs, adding both performance overhead and security risk

Chapter 6

System Architecture and Protocol Design

"*All models are wrong, but some are useful*". This well-known observation by George Box [217] reminds us that any system architecture is an abstraction—necessarily simplified and imperfect. Still, the goal of this chapter is to develop a model (architecture and protocol) that is useful: one that captures essential technical elements while remaining transparent enough to support critical evaluation.

Building on the motivation outlined in Section 1.4, we describe how Intel SGX can be used to securely perform statistical computations on data from multiple sources in a cloud environment. While commercial solutions for secure analytics exist (see Section 5.2), they typically operate as black boxes. Implementation details are proprietary and often subject to non-disclosure agreements, preventing independent evaluation of their security properties.

To address this, the system architecture presented is explicitly documented and grounded in known building blocks. At a high level, data owners (e.g., banks, hospitals) each provision sensitive data into their dedicated enclave running within an NSO application on cloud infrastructure, after verifying its authenticity through remote attestation. After initial pre-processing, this data is securely handed off to an NSO enclave for linkage. Final aggregation occurs in another NSO owned enclave, applying SDC before releasing results to the untrusted cloud environment.

A more detailed explanation is provided in the remaining parts of the chapter, structured as follows:

- **Section 6.1** introduces the system architecture, including design rationale, threat model, and conceptual layering.
- **Section 6.2** describes the protocol workflows that enable secure attestation, enclave-to-enclave communication, and data provisioning.
- **Section 6.3** analyzes platform considerations, trade-offs, and hardware support, motivating the choice of SGX.

6.1 System Architecture

The architecture is designed around a layered, defense-in-depth security model emphasizing isolation and compartmentalization. Sensitive data from each data owner is provisioned into dedicated enclaves hosted within the NSO cloud environment, limiting the potential impact of a compromise. By dividing the system into three conceptual layers—Data Ingestion, Linkage and Aggregation, and Output—the architecture separates distinct processing responsibilities, simplifying design analysis and reducing the TCB of each component. A more detailed walkthrough of these layers and their interactions is presented in Section 6.1.1.

Within this design, attestation plays a critical role. Before any sensitive data is provisioned, data owners must verify that the execution environment matches their security expectations. Each enclave is tailored to its specific function, minimizing unnecessary code that could expand the attack surface. SGX offer strong isolation guarantees in theory. However, they require careful design to avoid pitfalls such as bloated TCBs or complex interfaces, which can introduce vulnerabilities (see Section 7.2.1). Therefore, the architecture emphasizes minimal and well-defined interfaces recognizing that while perfect security is unattainable, a thoughtfully compartmentalized design can significantly reduce risks.

6.1.1 Architecture Overview

The system architecture is divided conceptually into three primary layers: **Data Ingestion**, **Linkage and Aggregation**, and **Output**. This layered design simplifies reasoning about the system by separating areas of concern, clarifying security boundaries, and isolating functionality into purpose-specific enclaves.

Figure 6.1 provides a high-level view of the components and information flows in the architecture. Only two data owners are illustrated to simplify the illustration, but architecturally more can be included, potentially adding more NSO enclave for linkage. Data owners provision their sensitive records into individual enclaves hosted in the NSO's cloud environment. Each data owner enclave processes only its respective dataset, applying initial data cleaning and transformation steps to prepare records for linkage while minimizing exposure of unnecessary personally identifiable information (PII). By compartmentalizing these steps, the design limits potential privacy breaches and reduces the TCB for each enclave.

Once enclaves are launched on the cloud platform, remote attestation becomes a crucial initial step. This process allows data owners to verify that the computing environment matches expected measurements (e.g., `MRENCLAVE`, `MRSIGNER`) and complies with security policies before releasing any sensitive data. The NSO's untrusted orchestration process is responsible for provisioning, launching, and maintaining these enclaves, but is never trusted with plaintext data.

After successful attestation and data provisioning, the linkage phase identifies records referring to the same entities across multiple data owners. This phase involves comparing QIDs inside secure enclaves, which can occur in a single ag-

gregator enclave or across multiple dedicated enclaves. By structuring linkage processing into a separate layer, the architecture accommodates different record linkage strategies—from simple deterministic matching to more advanced PPRL protocols.

The final **Output** layer aggregates matched records and produces statistical results. Sensitive intermediate results remain within enclaves throughout the pipeline. The final aggregation enclave can optionally apply SDC methods, such as differential privacy or noise injection, before releasing sanitized aggregate data. This ensures that sensitive raw records or individual-level information are never exposed outside the secure processing environment.

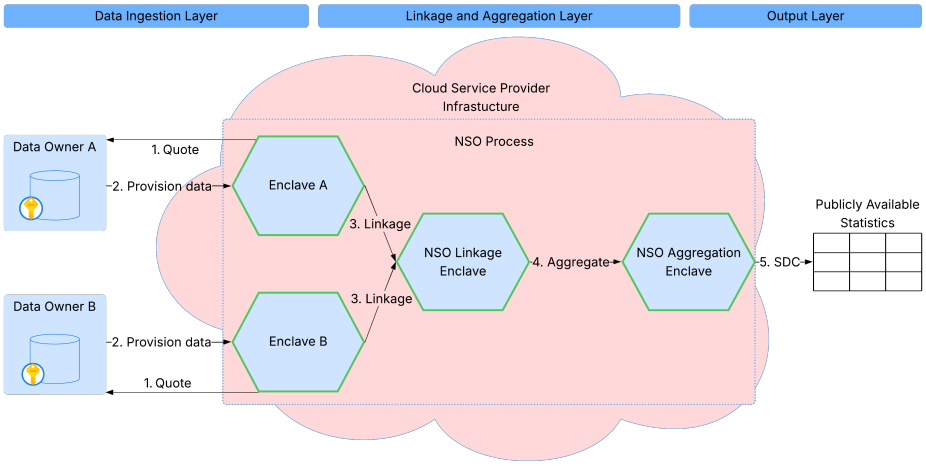


Figure 6.1: High-level overview of the system architecture and components involved. The division of functionality into enclaves is a design choice shaped by trade-offs between security requirements and development complexity, and can be adapted based on specific needs.

Although enclave development and distribution processes are not explicitly modeled here, they are critical to the system's trustworthiness. Decisions about what code and third-party libraries to include in enclaves must consider TCB growth carefully, recognizing that a bloated TCB increases the likelihood of vulnerabilities and complicates formal verification efforts. Literature on microkernel and unikernel approaches highlights how minimalism and careful dependency management can reduce TCB complexity and improve security [75, 206]. Automated tools for code analysis and continuous security assessment can assist in maintaining a manageable and auditable TCB over the system's lifetime.

Assuming data owners agree in advance on the schema of their datasets and the attributes required for linkage, the architecture can employ a simplified model where deterministic matching occurs inside a single aggregator enclave. However, if schema definitions differ between owners, or if linkage fields must remain masked even from the NSO, more sophisticated protocols, such as MPC or PPRL

using masked comparison techniques (e.g., Bloom filters), may be required. These decisions reflect trade-offs between system complexity, performance, and privacy guarantees.

6.1.2 Design Goals and Rationale

The architecture is guided by a set of design goals that reflect both the technical challenges of operating in untrusted cloud environments and the security requirements of statistical analysis.

First, the system adopts a **zero trust** stance toward the CSP, intermediary infrastructure, and even the orchestrating NSO process. Sensitive data must be protected from all parties except the enclave into which it is provisioned. This requires secure remote attestation, ensuring that data is only released into environments that meet the data owner's integrity expectations.

Second, the system aims to enable **confidential processing**, allowing computations over datasets from different data owners while keeping each party's input confidential until aggregate results are produced. This necessitates a processing model where data is never decrypted or exposed outside of the enclave responsible for handling it.

To reduce attack surfaces, the architecture emphasizes a **minimal TCB** within each enclave. This involves isolating responsibilities, stripping unnecessary dependencies, and avoiding shared enclave logic where possible. In addition, for auditability, trust in the behavior of each enclave is grounded in the ability to verify its codebase against known measurements. To support this, data owners and the NSO are expected to exchange source code and corresponding enclave measurements during deployment or onboarding. This allows data owners to audit the enclave logic ahead of time, and later verify at runtime that the enclave they are provisioning data to matches the vetted implementation.

The layers are separated, or compartmentalized, into distinct enclaves keeping with **defensive design principles**. This is meant to limit the blast radius, in the event that an enclave becomes compromised.

Finally, the system considers **asynchronous data processing**, allowing records or encrypted payloads to be staged, sealed, or handed off between enclaves without requiring synchronization between all data owners. This facilitates flexible workflows, particularly when data owners operate on different schedules or produce data at varying rates.

6.1.3 System Roles and Trust Model

The system architecture involves multiple stakeholders (illustrated in Figure 6.2), each with distinct roles and corresponding trust assumptions:

Data Owners Independent organizations or agencies that contribute sensitive datasets for record linkage and statistical analysis. Each data owner retains control over its data and is responsible for verifying the integrity of the processing

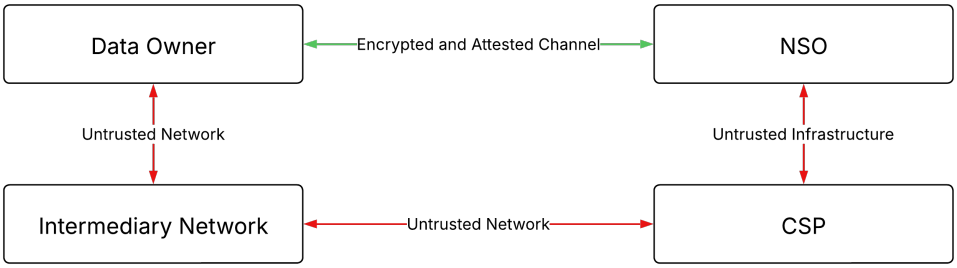


Figure 6.2: Logical overview of the system's trust relationships. Data Owners trust the integrity and confidentiality of enclave logic, which the NSO manages and Data Owners verify through remote attestation to establish secure communication. The CSP is not trusted with data confidentiality or integrity, relying instead on SGX's hardware-enforced isolation. Any intermediary network providers are outside the threat model, as their untrusted nature is mitigated by end-to-end encrypted communication.

environment through remote attestation before releasing any sensitive information.

Cloud Service Provider (CSP) An infrastructure provider responsible for hosting the NSO's enclave workloads. The CSP is treated as fully untrusted: while it provisions the hardware and runtime environment, the system design assumes that confidentiality must be preserved even in the presence of a compromised CSP. Hardware-based TEEs are used to enforce this isolation.

National Statistics Office (NSO) A coordinating entity responsible for managing the record linkage and aggregation process. Although the NSO operates the enclave infrastructure and orchestrates workflows, it is not trusted to access raw input data. Trust in the enclave logic is established through attestation rather than institutional trust in the NSO itself.

Intermediary Infrastructure Providers Network operators or third-party services involved in transmitting data between data owners and the NSO's environment. These entities are considered out-of-scope for the threat model. Since all traffic is encrypted end-to-end, compromise of intermediary infrastructure is assumed to have no impact on data confidentiality or integrity.

6.1.4 Threat Model

It is necessary to note that defining an effective threat model requires balancing theoretical risks with practical realities. Security is inherently a trade-off involving performance, complexity, and resource constraints. While aiming for maximum security may sound ideal, it is often impractical. For instance, although microarchitectural side-channel attacks on TEEs have been demonstrated in research, there are no widely publicized real-world attacks compromising deployed TEEs through these techniques.

This architecture assumes a powerful adversary consistent with the standard

TEE attacker model [16], where attackers have full control over all untrusted system software. While enclaves protect confidentiality, they cannot prevent a malicious data owner from submitting malformed or strategically crafted input. This introduces the possibility of adversarial contributions that might distort linkage results or attempt to manipulate aggregate statistics. To help detect such behavior, enclaves emit metadata that can be inspected during or after processing.

6.1.5 Security Guarantees

The system is designed to mitigate the threats described above while recognizing the inherent limitations of TEE-based security. Its primary objective is to ensure **enclave confidentiality**, such that no privileged software adversary—including a compromised OS or hypervisor—can access data within an enclave. To further strengthen isolation, the architecture supports **enclave-to-enclave separation**, ensuring that the compromise of one enclave cannot affect the confidentiality or integrity of others.

A critical part of the design is the use of **minimal interfaces**, which reduce the attack surface exposed to adversarial input. The system is also designed with **resilience in mind**: if a vulnerability is discovered, TCB recovery [1] mechanisms allow enclaves to be patched and revalidated. These goals collectively support a robust enclave model suitable for multi-party statistical processing in untrusted cloud environments.

6.1.6 Design Specifications

A central objective of this architecture is to compartmentalize data provisioning: each data owner's sensitive input is handled within a dedicated enclave. By isolating data in this way, the potential impact of a breach is limited to a single provider's enclave, aligning with the defense-in-depth principle[2]. This approach creates multiple layers of security mechanisms to protect against a wide range of threats.

The value of compartmentalization has been demonstrated in other domains, such as browser sandboxing frameworks like Chrome's NaCl, which isolate web content to contain potential exploits, and operating system architectures like Qubes OS [218], which rely on strong inter-compartment boundaries to protect sensitive workloads. Projects like Gramine [212] have shown how similar principles can be applied to TEEs, enabling the secure execution of unmodified applications in enclaves.

However, the benefits of compartmentalization come with challenges. Securely designing interfaces between compartments is difficult. Complex message parsing and marshalling can introduce new attack surfaces. Care must be taken during implementation to define narrow, well-structured interfaces that reduce parsing complexity and limit opportunities for exploitation.

[1] https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/best-practices/trusted-computing-base-recovery.html

[2] NIST definition: https://csrc.nist.gov/glossary/term/defense_in_depth

Constant-time programming techniques are important to reduce susceptibility to timing side-channel attacks, and memory-safe languages like Rust could further reduce risks from memory corruption vulnerabilities.

Moreover, beyond simply isolating computations, enclaves can embed data usage and processing policies directly into their logic. By verifying attestation measurements, data owners can be assured that only enclaves enforcing approved policies will process their data, reducing dependence on external contractual agreements.

6.1.7 Limitations and Assumptions

The threat model described above operates under several important assumptions and limitations. First, SGX does not provide protection against side-channel attacks, which is why constant-time programming is advised when possible and applicable.

Second, direct physical attacks are considered out of scope, against consistent with the SGX threat model. These attacks require substantial hardware access and specialized capabilities that fall outside the adversary model targeted by this system.

Finally, the system assumes that data harvesting, cleaning, and preparation are performed externally by the data owners. Tasks such as data selection, schema alignment, and secure pre-transmission encryption are presumed to occur "out-of-band," before any interaction with the TEE-based platform. As such, the integrity and correctness of input data are not enforced by the system architecture itself.

6.2 Protocol Design and TEE Integration

In this section, we detail how SGX is used to implement the layers described in Section 6.1.1. The goal is not only to demonstrate where TEE mechanisms are integrated, but also to explain how they are concretely applied to achieve end-to-end confidentiality, attestation, and controlled data flow. Each subsection corresponds to a logical layer in the architecture and highlights the relevant security guarantees, system design choices, and TEE-specific operations.

6.2.1 Data Ingestion Layer

The data ingestion layer is responsible for establishing trust between the data owner and the cloud-hosted SGX enclaves before any sensitive data is released. This includes remote attestation of the enclave, secure provisioning of secrets, and the initial processing of records. Trust establishment in this layer is critical: it forms the basis for all subsequent secure computations, as any compromise at this point would undermine the system's confidentiality guarantees.

Remote Attestation

As discussed in Section 3.3.2, Intel SGX supports remote attestation by generating a cryptographically signed quote that captures enclave identity information such as `MRENCLAVE`, `MRSIGNER`, and security-relevant attributes. Section 3.3.2 focused on how this capability is grounded in hardware trust, with the QE signing locally generated reports to produce quotes for external verification. This section explains how these mechanisms are integrated into the system architecture using Intel's Data Center Attestation Primitives (DCAP), which provide an IAS-independent framework for quote generation and verification.

DCAP is Intel's recommended model for ECDSA-based attestation in cloud and data center environments. Unlike the legacy EPID-based model, DCAP does not require a live connection to Intel's Attestation Service (IAS). Instead, it allows quote generation and verification to be performed locally, using a flexible provisioning model and a standard certificate chain rooted in Intel's trusted root CA [219, 220]. Since IAS was retired in April 2025[3], this model is increasingly relevant.

DCAP divides attestation into two operational stages: **deployment** and **runtime**.

Deployment During deployment, the infrastructure required for remote attestation must be set up. The SGX-enabled platform is first configured with SGX enabled in its UEFI or BIOS settings [203, 204]. This ensures that architectural enclaves such as the Quoting Enclave (QE) and the Provisioning Certification Enclave (PCE) can operate correctly. The PCE acts as a local certificate authority that certifies the attestation key used by the QE, producing a certificate chain rooted in a device-specific Provisioning Certification Key (PCK) [219, 221].

To obtain the PCK certificate, the platform uses the PCK Certificate ID Retrieval Tool to collect hardware identifiers, including CPU IDs, SVN, and TCB metadata. This information is signed by the platform and submitted to Intel's Provisioning Certification Service (PCS), which verifies the data and issues the corresponding PCK certificate [204, 219]. The platform is then registered with the Provisioning Certificate Caching Service (PCCS), which downloads and stores this certificate along with other required collateral such as certificate revocation lists (CRLs), the signing chain, and TCB information [222]. These artifacts are cached locally and can be reused across multiple attestation sessions without requiring live internet access.

Starting with 3rd Gen Xeon Scalable processors, operators may optionally use a custom attestation key infrastructure instead of Intel's default PCS-based model. This allows attestation keys to be provisioned under an operator-controlled chain of trust, enabling private attestation infrastructures for cloud or on-premise deployments [85]. The use of long-lived PCK certificates ensures that once provisioning is complete, attestation can proceed even in disconnected or isolated environments [223].

[3] `https://community.intel.com/t5/Intel-Software-Guard-Extensions/IAS-End-of-Life-Announcement/td-p/1545831`

Runtime Once deployment is complete, the platform is ready to generate attestation quotes.

Quote Generation. The QE produces a quote, signing it with the cached PCK certificate. A challenge nonce from the verifier is typically included in the quote generation process to guarantee freshness and protect against replay attacks [221, 222]. The quote securely conveys the enclave's identity and integrity measurements, including `MRENCLAVE` and other attributes [221, 222].

Quote Verification. The quote, along with the cached collateral, is transmitted to the verifier (data owner), who uses Intel's DCAP quote verification library to validate it. The verifier uses the PCK Signing Chain, rooted in Intel's trusted root CA, to authenticate the QE's attestation key and validate the quote's signature [203, 219]. Verification checks include signature chain integrity, CRL status, TCB level, and that enclave measurements match expected values [203, 222]. Verification can be performed inside an enclave (trusted verification) or outside (untrusted verification), depending on the data owner's architecture.

Several practical code samples from Intel[4] and OpenEnclave[5] corroborate this DCAP verification model. They showcase embedding DCAP ECDSA quotes into TLS certificates (RA-TLS), integrating attestation into the TLS handshake without requiring protocol changes [223, 224].

A key feature of DCAP is flexibility in collateral retrieval. Although it is typical for the attester to send collateral alongside the quote, data owners outside the cloud environment can directly query Intel PCS or their own PCCS instance to independently fetch collateral (e.g., updated CRLs or TCB data) [221, 222]. This decouples trust establishment from the attester's runtime environment and enables independent verification workflows.

While Intel provides reference implementations of the QE and the DCAP libraries, third parties can implement their own QEs and attestation infrastructures, allowing organizations to build or customize their attestation flows, for example by writing their own QE or PCCS implementations [219, 221].

Table 6.1 summarizes the key roles and responsibilities within this DCAP-based attestation model.

Attestation Flow Overview. Figure 6.3 illustrates the DCAP-based remote attestation process. While ①, where the enclave requests data from the data owner, is not formally part of the attestation protocol, it is included to reflect how attestation is commonly triggered in real-world systems. ② The data owner responds with a challenge (typically a nonce) to verify the integrity and freshness of the requesting enclave. ③a The platform retrieves the PCK certificate and attestation collateral from the PCCS, if not already cached. ③b The enclave generates a REPORT structure containing its measurements and passes it to the platform's QE. The QE verifies the report and produces a signed quote that binds the enclave's

[4]Intel Confidential Computing Zoo Github repository: `https://github.com/intel/confidential-computing-zoo`

[5]OpenEnclave Github repository: `https://www.github.com/openenclave/openenclave`

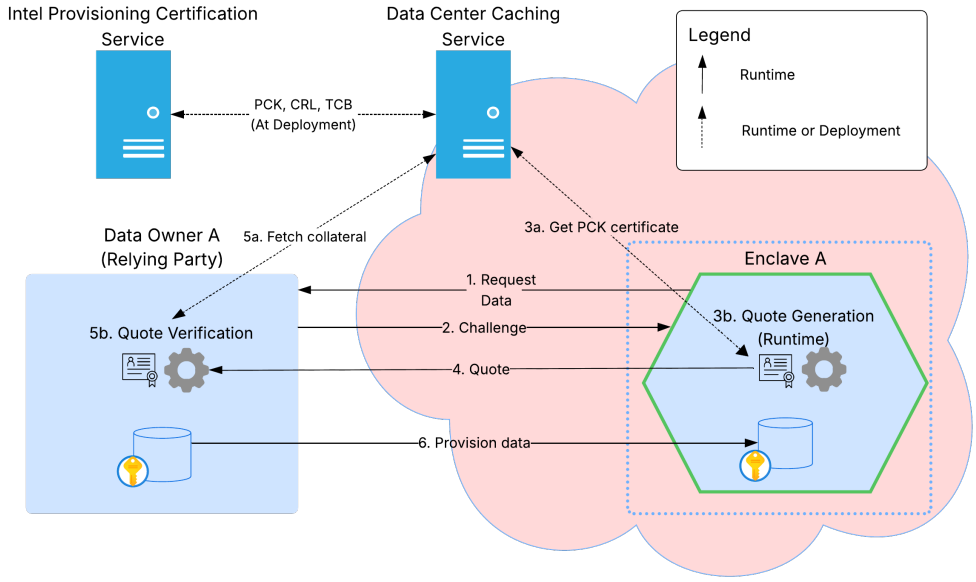


Figure 6.3: Illustration of DCAP attestation flow showing quote generation, collateral retrieval, and quote verification steps. Solid lines represent runtime activities; dashed lines indicate operations performed at deployment or optionally at runtime.

Table 6.1: Roles and responsibilities in DCAP-based attestation

Role	Entity in Model	Responsibility
Challenger	Data Owner	Initiates attestation by requesting a quote from the enclave, optionally including a nonce for freshness.
Verifier / Relying Party	Data Owner	Validates the quote and collateral using Intel's DCAP verification library; assesses TCB level, `MRENCLAVE`, and other enclave attributes; decides on provisioning secrets.
Attester	SGX enclave	Generates quotes using the QE, gathers collateral from PCCS, and transmits these to the Data Owner.
Provisioning Certificate Caching Service (PCCS)	Operated by NSO or CSP	Supplies cached attestation collateral (PCK certificates, CRLs, TCB information) to the attester or verifier, reducing dependence on live access to Intel PCS.

identity to an attestation key certified by the PCK. ④ The quote is returned to the data owner application. ⑤a The data owner may retrieve attestation collateral from a Provisioning Certificate Caching Service (PCCS, or data center caching service in the figure), either operated by Intel or hosted independently. This includes the PCK certificate chain, CRLs, and TCB information required for quote verification. ⑤b The data owner verifies the quote using Intel's DCAP quote verification library. This involves checking the certificate chain, revocation status, TCB level, and enclave identity values such as `MRENCLAVE`. If the verification is successful, the verifier can continue with secure provisioning.

⑥ The final provisioning step is described in the next section.

While this section outlines the attestation process at a protocol level, practical implementations can benefit from existing frameworks that automate parts of this workflow. For example, Gramine [225] supports DCAP-based attestation and offers tooling to simplify deployment, including support for RA-TLS. These frameworks can ease deployment without altering the protocol-level design described here. Development considerations are discussed in Section 5.3.1, while the security implications of using shielding runtimes are analyzed in Section 7.2.1.

Provisioning Sensitive Data

Before provisioning any sensitive data, the verifier must enforce security policies based on the attestation results. This involves checking enclave identity measurements (e.g., `MRENCLAVE`, `MRSIGNER`) and attributes (e.g., ensuring the enclave is not running in debug mode), as provisioning secrets to an untrusted or improperly configured enclave could compromise data confidentiality.

In addition to validating enclave measurements (e.g., `MRENCLAVE`), the verifier must have confidence in what those measurements represent. This requires that the enclave logic be known and vetted in advance, typically by linking hash measurements to reviewed binaries or published source code. In practical deployments, this might involve maintaining an allowlist of expected measurement values or relying on reproducible builds that enable independent verification. Such vetting ensures that enclaves enforce agreed-upon data processing policies before any sensitive input is provisioned. Without this step, a malicious or misconfigured enclave could exhibit unexpected behavior, even if its measurement passes verification. This requirement is particularly important in multi-party settings, where trust must be grounded in verifiable code rather than institutional reputation. Only after these policy checks pass should the relying party proceed with secure provisioning.

Provisioning itself typically involves establishing a secure channel that is cryptographically bound to the attestation evidence: after successful verification, the enclave's ephemeral public key—generated during or before quote generation and included in the attestation report—allows the relying party to encrypt a symmetric key or secret data so that only the enclave can decrypt it. This mechanism ensures secrets are provisioned exclusively to the attested enclave, even if the transport

channel is untrusted. Intel's SGX architecture directly supports this workflow [203, 204].

As a design decision, enclaves may seal secrets after provisioning, storing them encrypted and bound to the enclave's identity for reuse across restarts. Alternatively, secrets may be streamed, handled asynchronously, or exchanged via shared memory depending on the nature of data sources, some of which might produce continuous streams while others deliver periodic batches, and on the required reliability and freshness of the data. This flexibility allows attestation-driven systems to adapt provisioning and data handling strategies to specific workload and security requirements.

An alternative solution involves delegating secret provisioning to a cloud-based Key Management Service (KMS) that uses attestation evidence to enforce release policies. This would likely not be a piratical approach if the data owners is expected to stream data in real time. While such KMS-based approaches can streamline secret management and enable central auditing, they introduce an additional trust dependency on the KMS itself, which may also require attestation to assure relying parties of its integrity. A detailed analysis of KMS-based provisioning is therefore considered out of scope for this work.

6.2.2 Linkage and Aggregation Layer

Once enclaves have successfully attested to the data owners and been provisioned with their respective datasets, the next step is to establish trust and communication between each data owner enclave and the NSO's linkage enclave (see Step 3 in Figure 6.1). This phase enables secure coordination and computation across enclave boundaries, without exposing sensitive data to the untrusted environment.

Local Attestation and Enclave Communication in the NSO System

Local, or intra-platform, attestation is a prerequisite for secure communication between enclaves, as the identity and trustworthiness of the communication partner must be established before any sensitive data can be exchanged. Although this was described in Section 3.3.2, we provide a practical description of the steps involved here. The outline of this process is illustrated in Figure 6.4

When an enclave requests attestation, a special structure called a *REPORT* is generated by the hardware. This report includes the `MRENCLAVE` value (a hash representing the enclave's initial code and data), security-relevant state, and user-defined data. Crucially, the report is authenticated using a keyed MAC that only the target enclave can verify, by deriving a platform-specific `REPORT_KEY`. While the MAC itself is symmetric, the ability to derive this key is restricted to the target enclave through a hardware-internal asymmetric process. Although Intel's developer guide refers to this as a "signature block" over the data, it is not a digital signature in the traditional public-key sense, but a symmetric construct verified locally on the same platform [203, 226].

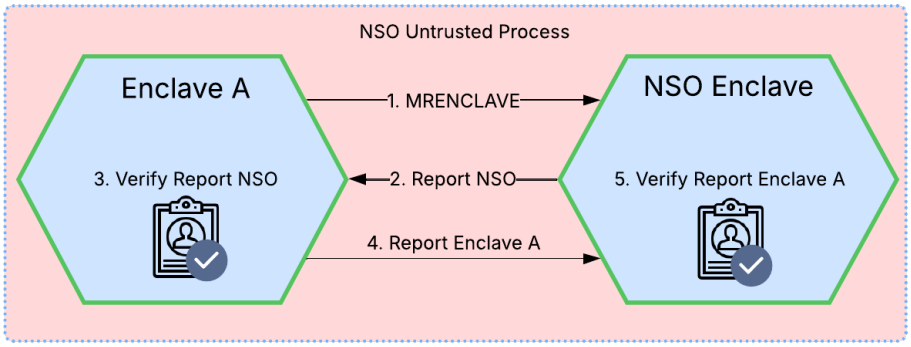


Figure 6.4: Local attestation between enclave A belonging to data owner A, and the NSO enclave.

The high-level goal in our case is to establish a secure channel between the data owner enclave and the NSO enclave so that sensitive linkage parameters or partial results can be exchanged without revealing them to the untrusted host or operating system.

Step 1

The NSO application retrieves the `MRENCLAVE` measurement of the data owner enclave. This can be done either by exporting it directly from the enclave or generating a dummy report for retrieval, as described in Intel's enclave-to-enclave communication whitepaper [226].

Step 2

The NSO enclave generates a `REPORT` destined for the data owner enclave. This report includes user-defined data—typically a Diffie-Hellman public key or similar—used to bootstrap a session key exchange. This report is passed through untrusted memory.

Step 3

The data owner enclave verifies the `REPORT` using the `EGETKEY` instruction to retrieve the platform-specific `REPORT_KEY` and validates the MAC. If the report is valid and targeted at this enclave, it proceeds to generate its own report and corresponding key exchange data.

Step 4

A similar process is then repeated in reverse: the data owner enclave sends a report and key exchange payload to the NSO enclave, completing the mutual attestation process and establishing a shared session key (also referred to as the AEK).

Step 5

With mutual attestation complete and a secure channel established, the enclaves can now exchange encrypted messages over untrusted memory. These messages include not just raw data, but also metadata such as call type, function ID, and input parameter lengths. This function ID marshalling mechanism (described in Intel's whitepaper [226]) makes it possible to implement enclave-to-enclave RPC-style calls over this secure channel.

Secure Data Handoff Between Enclaves. In scenarios where the data owner and NSO enclaves are developed by different parties, SGX sealing is generally not suitable for transferring data between them, since sealed blobs can only be unsealed by enclaves with matching sealing identities. Instead, encrypted handoff using the AEK provides a flexible alternative. After mutual local attestation, the data owner enclave can encrypt its output under the AEK and store it to disk or memory. The NSO enclave, once launched, can retrieve and decrypt this payload without requiring synchronous execution or shared sealing keys. This approach supports asynchronous workflows while maintaining end-to-end confidentiality.

Record Linkage and Aggregation

Now that data from multiple independent source are accessible to the NSO enclave, they have to be matched/linked. As mentioned previously, this can be challenging because it traditionally relies on comparing QIDs—attributes such as names, birthdates, or addresses—that are sensitive and cannot be exposed unprotected. Thus, a central question this system addresses is: *How can matching occur securely without revealing QIDs to untrusted parties or even to the orchestrating NSO?*

The system supports two main strategies for secure record linkage inside enclaves:

1. **Deterministic Linkage:** If data owners agree on a consistent schema and stable QID fields, deterministic matching can be performed within a single aggregator enclave hosted by the NSO. In this approach, enclaves receive raw or lightly transformed QIDs, execute exact matches, and release only aggregated results. While this simplifies the implementation, it requires trust in the aggregator enclave and careful schema alignment across all data owners.

2. **PPRL:** When QIDs cannot be revealed even to the aggregator enclave, advanced PPRL techniques, such as masked Bloom filter encoding, can be used [227]. Here, each data owner enclave computes masked representations of QIDs, ensuring raw identifiers never leave the originating enclave unmasked. The aggregator enclave, or a series of intermediate enclaves, then compares masked QIDs using approximate similarity metrics to identify probable matches while minimizing privacy risks.

To illustrate how PPRL integrates into the architecture, consider the standard phases [227]: *blocking* (grouping candidate pairs), *comparison* (computing similarities), and *classification* (deciding matches). Each phase can operate in its own enclave, compartmentalizing processing and limiting potential exposure. This aligns with the architecture's defense-in-depth approach and leverages secure enclave-to-enclave communication established via local attestation (see Section 6.2.1).

Figure 6.5 shows a generalized PPRL workflow, highlighting where enclaves can secure individual steps. While this design does not prescribe a specific linkage algorithm, it demonstrates how the architecture accommodates privacy-preserving linkage at scale and supports integration of advanced techniques described in recent TEE-PPRL research [26].

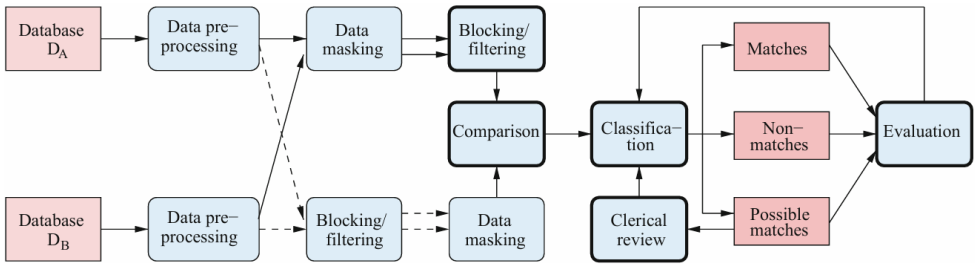


Figure 6.5: General steps in the PPRL process, with enclaves securing each phase. Figure from [227].

Finally, once matching is complete, the **aggregation phase** collects confirmed links and computes statistics such as counts or aggregates, all within an NSO-managed enclave. This ensures that individual-level matches or raw QIDs never leave the secure processing environment unprotected.

Metadata. To improve robustness against adversarial input from participating data owners, both the linkage and aggregation enclaves emit non-sensitive metadata during processing. This metadata may include metrics such as the number of matches per data source, linkage success rates, or schema validation summaries. While it does not reveal any sensitive input records, it enables the NSO to identify irregularities that could signal malformed or strategically crafted submissions. By inspecting this metadata, the NSO can detect potential manipulation or data inconsistencies without compromising the confidentiality of the underlying records.

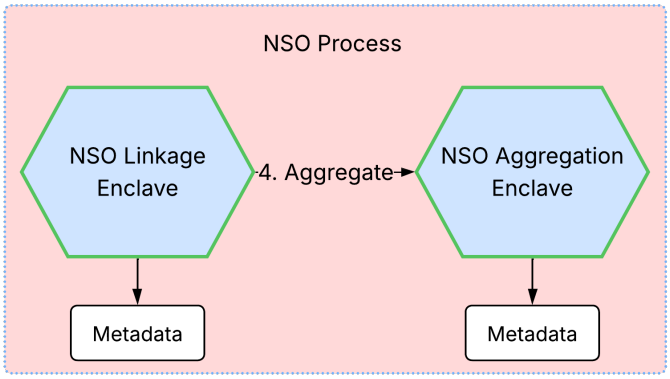


Figure 6.6: Metadata emission from linkage and aggregation enclaves enables post-hoc validation and anomaly detection without compromising confidentiality.

Several mechanisms are possible for emitting this metadata. It may be written to untrusted NSO memory, stored to disk, or sealed for later inspection by a dedicated "auditor" enclave. The specific mechanism is not specified, as it is considered an implementation-level detail.

6.2.3 Output Layer

The final enclave, which aggregates and computes statistical results, is central to providing output privacy. This complements the system's input privacy, which is enforced through TEEs and secure record linkage during earlier processing stages. Output privacy ensures that the information released after secure processing does not enable re-identification of individuals, even if aggregate data is disclosed publicly.

To address this, the design allows applying SDC methods directly inside the final enclave. SDC techniques include suppression, generalization, noise injection, or formal DP. Each method aims to limit what can be inferred about individual records from aggregate statistics. For example, DP provides mathematically proven guarantees by bounding an individual's influence on the released results, although it requires careful selection of privacy parameters like epsilon to balance privacy with accuracy.

This system architecture does not enforce a specific SDC method but supports integrating any post-processing strategy chosen by the NSO within the final enclave. By computing SDC or DP transformations inside the enclave, intermediate results such as cell counts or partial aggregates remain protected until sanitized data is finalized. This modularity enables NSOs to apply appropriate SDC measures tailored to their regulatory, statistical, and privacy requirements.

6.3 Choice of TEE

Selecting an appropriate TEE technology is central to implementing the proposed system's protocol securely and efficiently. Although many novel TEEs have been proposed in academic research, practical deployment is constrained to commercially supported offerings with mature tooling and cloud platform availability. Since the architecture aims to leverage cloud-based infrastructure, among the widely supported, this limits the viable options to Intel SGX, AMD SEV, and Intel TDX.

Intel SGX Capabilities and Suitability. Intel SGX provides process-level isolation by enabling enclaves that protect selected code and data regions within a process. This fine-grained protection supports the proposed design's goals of compartmentalization and minimizing the TCB. SGX supports *local attestation*, which allows enclaves on the same platform to establish mutually authenticated secure channels — an essential feature for securely linking data owner enclaves with the NSO aggregator. By enabling direct enclave-to-enclave Diffie-Hellman key exchanges, local attestation offers both security and performance advantages over channel establishment mechanisms that depend solely on external parties.

SGXv2 introduced Enclave Dynamic Memory Management (EDMM), allowing enclaves to dynamically allocate up to 512GB of protected memory, significantly improving scalability for memory-intensive applications. Importantly, Intel provides tooling and documentation for TCB recovery, allowing enclave deployments to remain secure over time despite vulnerabilities. If a flaw is discovered in the enclave logic or platform firmware, operators can revoke outdated attestation keys and re-deploy enclaves with updated measurements, preserving security without rebuilding the entire infrastructure [6].

Intel's own recommendations suggest SGX as the preferred TEE for new applications requiring strong security guarantees [228]. Although SGX has been the target of numerous academic attacks, its mature implementation and strong threat model against privileged software adversaries have made it a popular choice for security research, not necessarily a reflection of inherent weakness. As noted by Lee et al. [229], SGX was targeted because it offered the strongest security guarantees for data confidentiality even against compromised operating systems and hypervisors.

Considerations on VM-Based TEEs: SEV and TDX. In contrast to SGX's process-level protection, AMD SEV and Intel TDX offer VM-level memory encryption, allowing unmodified legacy applications to run securely inside encrypted virtual machines. This coarse granularity can simplify some deployments but is less compatible with the proposed architecture's need for isolating individual data owners' computations into separate enclaves.

[6] `https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/best-practices/trusted-computing-base-recovery.html`

A practical concern with VM-based TEEs is the operational complexity associated with rapidly evolving confidential VM technologies. Cloud providers may require frequent guest OS updates to maintain compatibility with evolving VM-based TEE implementations, introducing maintenance overhead [230]. Although emerging solutions like OpenHCL and paravisor frameworks aim to reduce this burden, they remain early in adoption.

Intel TDX, while promising in its focus on confidential VMs, currently does not support running SGX enclaves inside guest TDs, as stated in the TDX Module Base Architecture Specification v1.5: "Running Intel SGX enclaves within a guest TD is not supported" [98]. This incompatibility precludes combining SGX enclaves with TDX-based VMs, limiting TDX's suitability for architectures requiring enclave-level isolation.

Granularity of Protection. The protection granularity difference between SGX and VM-based TEEs has significant implications for architecture design. SGX allows developers to explicitly partition applications into trusted and untrusted components, minimizing each enclave's TCB. In contrast, SEV and TDX secure entire VMs, which can protect unmodified legacy workloads but cannot isolate fine-grained functions or individual data owners' contributions [209].

Deployment Target and Platform Availability. SGX is primarily featured on Intel commodity CPUs, with limited server-grade platform availability depending on the cloud provider. AMD SEV, integrated into EPYC processors, enjoys broader adoption among cloud service providers. Since this system targets deployment on cloud infrastructure to support multi-party statistical workloads, SGX2's availability must be verified explicitly for each potential deployment platform. Tools like `cpuid` can confirm support for essential SGX features such as Flexible Launch Control (FLC) and Key Separation and Sharing (KSS), which are important for dynamic enclave deployment (see Code listing 6.1).

Code listing 6.1: Checking supported SGX version on the system

```
adminsgx@vmSGX:~$ cpuid | grep -i sgx
      SGX: Software Guard Extensions supported = true
      SGX_LC: SGX launch config supported      = true
   Software Guard Extensions (SGX) capability (0x12/0):
      SGX1 supported                           = true
      SGX2 supported                           = false
```

Chapter Conclusion. Given the architectural need for fine-grained, enclave-level protection, mature developer tooling, and support for local attestation, Intel SGX is the most suitable TEE for implementing the proposed protocol. While AMD SEV and Intel TDX offer advantages for legacy application compatibility at the VM level, their granularity and current limitations do not align with the system's requirements for compartmentalized processing of data from multiple independent parties.

Chapter 7

Discussion

We now examine the practicality, limitations, and broader implications of deploying TEE-based systems for privacy-preserving statistical computation in untrusted cloud environments.

The remainder of the chapter is organized as follows. Section 7.1 examines core security limitations of TEEs, including microarchitectural vulnerabilities, design complexity, and evolving trust assumptions such as those introduced in SGX2. Section 7.2 addresses challenges in the software ecosystem and development lifecycle, including shielding runtimes, platform support, and long-term maintenance. Section 7.3 explores the relationship between privacy and trust in this context. Finally, Section 7.4 places the findings in a broader perspective, comparing TEEs with alternative PETs and reflecting on their long-term viability and societal relevance.

7.1 Security Analysis

Modern computing systems rely on a layered stack of components, forming abstractions that help developers focus on functionality rather than low-level implementation details. These abstractions are essential for managing complexity, but they come at a cost. As Ferguson et al. observe, "Complexity is the worst enemy of security, and it almost always comes in the form of features or options" [231]. The modern CPU, along with the TEE security model[1] built on top of it, has grown far beyond the tractable reasoning model of early computing systems. The Intel Software Developer's Manual now exceeds 5,000 pages [68]. For comparison, the *MOS Technology 6502* microprocessor from 1975 had around 3,500 transistors, fewer than the SDM has pages [232].

Despite their appeal as solution to securing *data in use*, TEEs remain vulnerable to deep-rooted issues in the underlying hardware. These challenges are not merely implementation bugs, but often stem from systemic design patterns in

[1]While TEEs are fundamentally hardware-backed, their design presents a conceptual security model for isolated computation.

modern processors. In this section we discuss three interrelated limitations that constrain the security of TEE-based systems: microarchitectural vulnerabilities, hardware design, and shifts in TEE design and threat models that alter the assumptions enclaves can safely rely on. Understanding these constraints is critical for NSOs and other stakeholders seeking to assess whether TEEs provide a sufficiently reliable and secure execution environment.

7.1.1 Fundamental Problems

As discussed in Section 3.6, microarchitectural and side-channel attacks continue to challenge the integrity of TEEs. These attacks exploit leakage channels not due to logic errors in enclave code, but due to shared resources and performance optimizations inherent to modern processor designs. For this reason, they have been described as TEEs' "Achilles heel" [127]. Some research has even stated that "Intel SGX is Not the Answer" [233] due to this limitation.

Modern CPUs expose a range of subtle interactions through shared microarchitectural components such as caches, branch predictors, internal buffers, and execution pipelines. These resources are often reused across processes and privilege levels. Even when memory access controls enforce strict boundaries, the timing or state of shared structures can leak fine-grained information about a victim enclave's internal behavior. Researchers have proposed that many of these vulnerabilities resemble use-after-free bugs in transient structures like the store buffer or fill buffers [145], illustrating their deep entanglement with speculative execution.

Attacks such as Meltdown and the MDS family demonstrate how speculative execution can transiently bypass architectural protections, allowing adversaries to extract secrets across isolation boundaries. Spectre-style attacks, on the other hand, exploit mispredicted control flow and speculative access patterns to encode secrets into observable microarchitectural state. These attack classes have not only proven difficult to fully mitigate, but also expanded our understanding of what constitutes architectural leakage.

This problem is far from new. Lampson's classic "confinement problem" [60] articulated the fundamental challenge of preventing information leakage in shared computing environments. The U.S. Department of Defense's evaluation of what constitutes a "Trusted Computer System" further reinforced this concern, establishing concrete thresholds for covert channel bandwidths in secure systems [234]:

> "Therefore, a Trusted Computing Base should provide, wherever possible, the capability to audit the use of covert channel mechanisms with bandwidths that may exceed a rate of one (1) bit in ten (10) seconds."

While such criteria may seem outdated, they underscore that covert channels have long been recognized as critical obstacles to trustworthy system design. As Rutkowska observed, "It's generally believed that it is not possible to eliminate

covert channels on x86 architecture, at least not if the system software was to make use of the multi-core and/or multi-thread features of the processors" [69].

Moreover, the statistical reliability of side-channel attacks is often underestimated. Even high-noise channels can be amplified through repeated measurement, exploiting the law of large numbers to extract secrets with high confidence. Practical exploitability is a function of both bandwidth and error tolerance, not just precision. This means even seemingly innocent leakage can cause serious breaches in confidentiality.

Despite numerous proposals for hardware-based defenses, most TEEs continue to rely on shared resources that expose leakage. Techniques randomized caches [138], Compiler-based solutions[235], or constant-time programming can reduce exposure, but do not eliminate leakage entirely. Crucially, many defenses remain incompatible with legacy software ecosystems or require privileged cooperation from the platform.

TEE vendors themselves reflect these limitations in their threat models. Intel's SGX documentation explicitly states that "SGX does not provide explicit protection from side-channel attacks" [203], and places the burden on enclave developers. Academic surveys echo this reality: "most (commercial) TEE proposals explicitly exclude side-channel attacks from their attacker model and recommend using existing countermeasures" [16]. At the same time, the attack surface continues to grow. Rubicon [236] and related work have shown that previously theoretical channels can become practical in real-world deployments. The opaque nature of modern CPU internals—including undocumented buffers and proprietary microcode—further complicates validation efforts. Without transparency, even expert developers cannot confidently assess whether isolation guarantees hold under adversarial conditions.

Mitigations are rarely without cost. Initial protections against Meltdown (KPTI) and Spectre (retpoline) degraded performance by approximately 2-11% [237]. More recently, the software defense against LVI introduced overheads ranging from 2x to 19x, depending on the workload [238]. The precise impact of these mitigations varies significantly depending on the application type, workload, and specific system. Collectively, these examples underscore the substantial performance cost of responding to attacks that ironically stem from aggressive performance optimizations in modern processor architectures. This inherent trade-off raises an important question: *at what point does the performance-security analysis deem security to be "too expensive"?* Disabling SMT would stop some a number of attacks, but is usually not done in practice for precisely this reason.

Finally, the pace of vulnerability discovery contributes to operational uncertainty. Critical side-channel vulnerabilities often persist across hardware generations, and vendor update cycles can leave systems exposed for months [121]. For NSOs and similar institutions, this creates a moving target in threat modeling. Ultimately, these side-channel challenges are not only technical, but also epistemic: they represent a gap between what TEEs claim to protect and what system owners can independently verify. For institutions like NSOs, this demands cautious

trust, layered mitigations, and clear boundaries on what TEEs can, and cannot, guarantee.

7.1.2 Hardware Design Complexity

The vulnerabilities described in the previous section may prompt a natural question: *Why have these issues not been resolved at the hardware level?* The persistence of microarchitectural flaws in commodity CPUs is not simply the result of engineering oversight, but reflects the inherent difficulty of designing and verifying complex hardware systems.

As shown in the HardFails study by Weisse et al. [239], many security-relevant bugs originate at the register-transfer level (RTL), where processor logic is implemented in Verilog or VHDL. These subtle flaws often escape detection because formal verification tools struggle to express and reason about global timing flows, speculative execution paths, internal cache states, and side effects on microarchitectural state. Instead, verification tends to focus on isolated modules, missing issues that arise from complex interactions between components. This limitation is particularly relevant for TEEs like Intel SGX or TDX, which rely not only on documented architectural protections, but also on implicit assumptions about internal behaviors such as speculative execution and buffer reuse. Many transient execution attacks exploit precisely the kinds of interactions that lie beyond what current formal verification approaches can fully cover.

Beyond these fundamental formal limitations, most hardware validation workflows still rely heavily on simulation and manual inspection. While necessary, these methods offer limited coverage for complex security properties, especially those related to timing and cache interactions. As a result, many vulnerabilities are only discovered post-deployment, through external research rather than comprehensive vendor testing. These persistent gaps create a disconnect between the security guarantees developers expect and the actual properties of deployed hardware.

These verification challenges extend significantly into the platform's lower software layers. Firmware and microcode, which are often proprietary and opaque, form part of the TCB but remain difficult to audit or reason about formally. Vulnerabilities in these layers have led to practical TEE breakage. For example, AMD SEV's remote attestation was shown to be vulnerable due to firmware-level issues in the AMD-SP component [111], while bugs in Intel's microcode update logic have directly impacted SGX integrity [240]. Microcode itself has been reverse-engineered and experimentally modified [241, 242], demonstrating that it is neither immutable nor fully trustworthy.

Building upon these challenges, the integrity of a TEE's remote attestation crucially depends on the trustworthiness of its boot trust anchoring. If the chain of trust starts too late in the boot sequence, or relies on mutable components, it may be possible to tamper with early components and still present a valid attestation. Although not in a TEE-based setting, BIOS Chronomancy [243] demonstrated this

by forging the CRTM, allowing attackers to produce seemingly valid boot records. In another example, Ermolov's research on Intel CSME [244] showed that provisioning keys for SGX could be recovered from certain legacy platforms via firmware vulnerabilities, undermining the integrity of the enclave identity and provisioning process. While Intel clarified that these specific attacks required physical access and were mitigated in newer chips [245], the broader implication remains: the root of trust is not invulnerable.

Finally, when considering the supply chain from an adversarial perspective, even RTL-level trust cannot be assumed. The "Analog Malicious Hardware" study [246] showed that attackers could embed capacitor-based analog circuits into unused chip layout regions. These implants could remain dormant during testing and activate only under highly specific runtime conditions, leading to privilege escalation. Such attacks evade both logic-level simulation and functional testing, illustrating a deep trust gap in chip fabrication pipelines.

In attempting to provide confidentiality, integrity, and availability on commodity hardware, TEEs may be overextending their practical scope. Although much of the system software is excluded from the TCB, there remains significant and often un-verifiable complexity in the hardware and firmware layers that form its foundation.

7.1.3 Shifting Trust Models (SGX2)

The second generation of Intel SGX, often referred to as SGX2, introduces new hardware capabilities, most notably dynamic memory management for enclaves. This transition coincides with Intel's strategic shift, discontinuing SGX support on client CPUs around 2022 and moving it exclusively to server-class processors like Ice Lake and Sapphire Rapids Xeons. While this shift aims to facilitate broader adoption and offer greater operational flexibility, it comes with significant changes to the underlying security guarantees, representing a recalibration of Intel's security posture for TEEs.

The most critical change in SGX's implementation on these server platforms is the removal or significant weakening of hardware-enforced memory integrity protection. Previously, the Merkle tree-based MEE provided robust integrity protection for enclave memory against physical attacks. With SGX on Ice Lake CPUs, this MEE was entirely removed, meaning hardware-level memory bus man-in-the-middle attacks are no longer mitigated. While Sapphire Rapids reintroduces partial integrity checking, it does so with a lightweight 28-bit MAC per cache line, which offers limited assurance [247]. This design choice also involved a transition from MEE to AES-XTS for memory encryption. While AES-XTS combines block encryption with address-based tweaks to maintain confidentiality, it explicitly does not include an integrity check. Consequently, attacks like replay or tampering at the memory bus level remain possible if a physical MitM is present.

Intel's decision reflects an updated threat model that either excludes physical DRAM attacks as a primary concern or assumes these can be addressed through

other means, such as robust data center physical security controls. Official records confirm that “Scalable SGX” (i.e., SGX2 as implemented in Xeon processors) explicitly does not provide memory integrity protection against a physical adversary [248].

Overall, SGX2 offers greater operational flexibility and targets a specific server deployment model, but at the explicit cost of weakening guarantees against physical memory attacks. This aligns its effective threat model closer to other server-focused TEEs like AMD SEV-SNP, which also operate under different assumptions regarding physical adversaries.

7.2 Software Perspective

While TEEs like SGX provide a hardware-based foundation for secure computation, their real-world deployment hinges on complex software stacks that bridge the gap between enclaves and the broader system. This includes SDKs, shielding runtimes, library OS frameworks, build environments, attestation services, and patching infrastructure. Each component introduces new responsibilities, interfaces, and potential vulnerabilities. For NSOs, which typically operate under strict trust, accountability, and auditability requirements, understanding these software-level dependencies is crucial for maintaining security over time.

This section examines the challenges associated with enclave development and integration, the role of shielding runtimes in enforcing security boundaries, the risks introduced by compromised development platforms or key material, and the practical burden of lifecycle management. These issues are central to the viability of the system proposed in Chapter 6, and directly influence whether NSOs can securely maintain enclave-based deployments across multiple years or hardware generations.

7.2.1 Vulnerabilities at the Edges

The interaction between enclaves after the local attestation (see Section 6.2.2), and between the NSO process and enclaves in general, are important considerations. From a security perspective, these "edges" are of particular interest, and vulnerabilities can often arise due to wrongful assumptions or incorrect implementations.

Modeling system security via privilege levels, as seen in Intel’s privilege rings, offers a useful abstraction, but one that fails to capture the practical realities of running enclave applications on real hardware. In practice, there is no clean separation between secure and insecure execution contexts. The enclave boundary is defined by hardware, but traversed by complex software. This has led to the rise of *shielding runtimes*, which aim to mediate that boundary, provide a secure bridge for enclave entry and exit, and alleviate some of burden put on developers.

TEEs are often conceptually compared to a "secure oasis", offering a protected environment within a hostile system. This is precisely the reason they are referred

to as *enclaves*. but most enclave applications are not written with this isolation boundary in mind. Developers depend on SDKs or LibOS-based runtimes that handle the messy parts of transitioning between trusted and untrusted code. As shown in Figure 7.1, this transition cuts across multiple layers: from the enclave application, through the API and ABI layers, and down to the shim layer and syscall interface. Each of these layers is a potential point of failure, especially since they are responsible for safely handling attacker-controlled input. As seen in other areas of security—as SSL/TLS certificate validation in SDKs and middleware—adversarial testing, unsafe library defaults, confusing APIs, and hidden failures in deep layers can completely undermine the security of otherwise sound protocols [249].

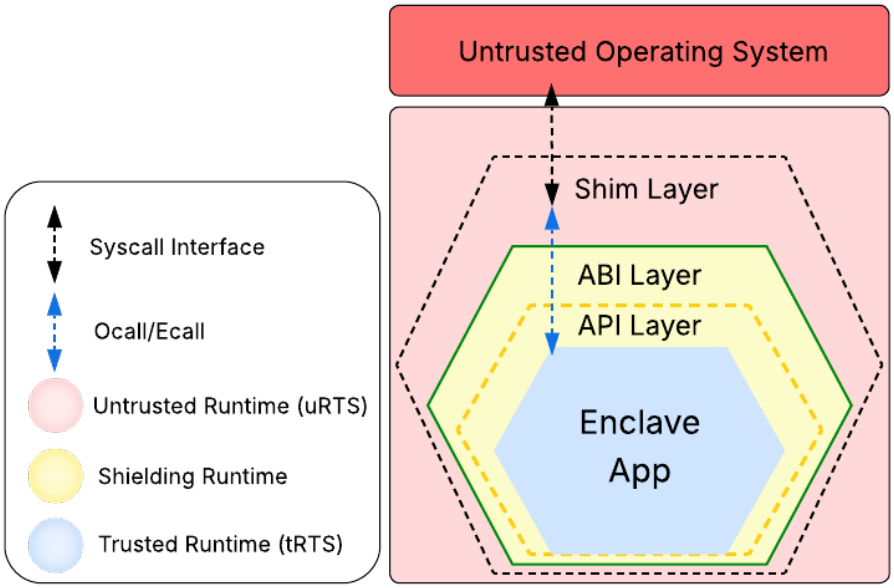


Figure 7.1: Typical layering of shielding runtimes and their interaction with enclave transitions. The shim layer (often referred to as edge routines in the documentation [203]) is depicted here outside the enclave boundary, where it usually resides to marshal and sanitize parameters. However, parts of edge routines may also exist inside the enclave as trusted edge code depending on the specific shielding runtime design.

LibOS and the Expanding TCB There are tradeoffs that must be considered with the use of LibOS-based approaches like Gramine. They offer convenience by handling system calls and memory abstractions, but at the cost of significantly increasing the TCB. This goes against one of the core motivations for using TEEs in the first place: reducing the attack surface. Extending it further and considering the case of a LibOS, which brings in a large amount of code that must now be trusted, audited, and updated. This is a practical aspect that is often downplayed

or overlooked. It is worth clarifying that while all LibOSes include a shielding runtime, not all shielding runtimes are LibOSes. This distinction is important, especially when assessing how much additional code is included in the TCB.

As recent studies show, LibOSs like Gramine, Occlum, and Mystikos reintroduce large syscall interfaces (often supporting over 100 syscalls)[16, 70], making them much larger than purpose-built SDKs like Fortanix EDP. This is logical when considering their purpose, which is to enable broader application compatibility. However, this convenience must be weighed against the enlarged TCB, a core contradiction in TEE philosophy.

tRTS and uRTS. The boundary between the trusted runtime system (tRTS) and the untrusted runtime system (uRTS) has been the cause of many vulnerabilities. Common issues include improper pointer validation, register state leakage, or inconsistent enclave resume behavior. Research has repeatedly shown that subtle oversights in these transitions can be exploited. Van Bulck et al. [250] revealed how ABI-level sanitization was inconsistently implemented across SGX runtimes. Specifically, this included aspects of such as register clearing and stack initialization. These bugs even affected Intel's own SDK and architectural enclaves like the QE, which plays a critical role in SGX remote attestation. The implication is that attackers can target the SDKs or shielding runtimes with relatively conventional software exploitation techniques, without needing advanced microarchitectural attacks. Studies [251] show that as of 2024, many SDKs still fail to fully sanitize CPU state or handle edge cases during ECalls and OCalls, leading to exploitable inconsistencies. New classes of attacks show that even asynchronous signal handling (e.g., injected by the OS) can be abused to violate enclave state integrity [179]. This implies the shielding runtime must treat *all interactions* with the untrusted environment, even error handling paths, as attacker-controlled. Techniques like in-enclave sanity checks on context and strict control-flow mediation are becoming necessary defenses.

Besides machine-state issues, shielding runtimes have been shown to reintroduce classical user/kernel-style bugs. Examples include pointer dereferencing before bounds validation, unguarded string length computation on untrusted memory, and time-of-check-to-time-of-use vulnerabilities. These often stem from incorrect assumptions that traditional user-mode code makes, *assumptions that fail when the adversary controls the host OS*.

The implication is that attackers can target the SDKs or shielding runtimes with relatively conventional software exploitation techniques, without needing advanced microarchitectural attacks. From a threat modeling perspective, this shifts the risk surface in an important way. Exploiting SDK-level vulnerabilities is more accessible to most attackers than building a microarchitectural exploit, and requires fewer preconditions.

ABI vs. API Sanitization

One particularly useful framing, highlighted by the research in [252], is the distinction between ABI-level and API-level sanitization. At the ABI level, details of the calling convention become important considerations. The shielding runtime must sanitize the machine state, including registers, flags, and stack pointers, to match compiler expectations. For instance, [250] found that several runtimes failed to set the Direction Flag, leading to a problem that is illustrated in Figure 7.2. Such errors may seem trivial, but Intel SGX is based on the x86 instruction set architecture (ISA), which follows the Complex Instruction Set Computer (CISC) design. This means it includes a large number of instructions and intricate combinations of flags and registers that influence execution behavior. These are meant to be useful and ensure that common operations are more efficient, but as the complexity of the underlying architecture grows, so does the attack surface. These ABI related details are low-level, language-agnostic, and often opaque to developers, but still an important aspect to be aware of when evaluating different runtimes and libraryOS frameworks.

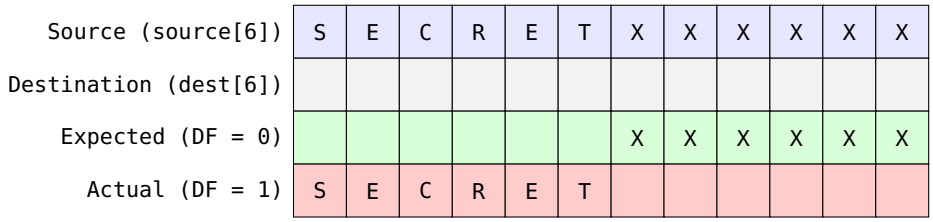


Figure 7.2: Intended copy (DF = 0) of the trailing `XXXXXX` region. If DF is maliciously set from outside the enclave, the copy runs in reverse inside the enclave and incorrectly copies the leading `SECRET` region into the same destination memory range.

At the API level, the shielding runtime must validate untrusted inputs like pointers, buffers, strings, and structs before use by enclave code. This includes checking that pointers reference untrusted memory regions only (see Figure 7.3 for how confidentiality can be breached when this is not done properly), that variable-length buffers do not overflow, and that inputs like strings are well-formed and null-terminated. These checks are harder, more language-specific, and prone to silent failure. For instance, failing to validate that a pointer lies outside of enclave memory can turn a simple echo server into a confused deputy that leaks enclave secrets. Moreover, certain checks like computing string length before validating the pointer can introduce subtle side channels or time-of-check-to-time-of-use (TOCTOU) bugs.

As a developer, this split is critical to understand: even if your application logic appears correct at the API level, it may still be vulnerable if underlying ABI-level assumptions are violated or the checks are insufficient, and vice versa. The *specific* examples highlighted above have been patched by all of the major runtimes and SDKs. Being aware of these security concerns is still important, as it is an attack

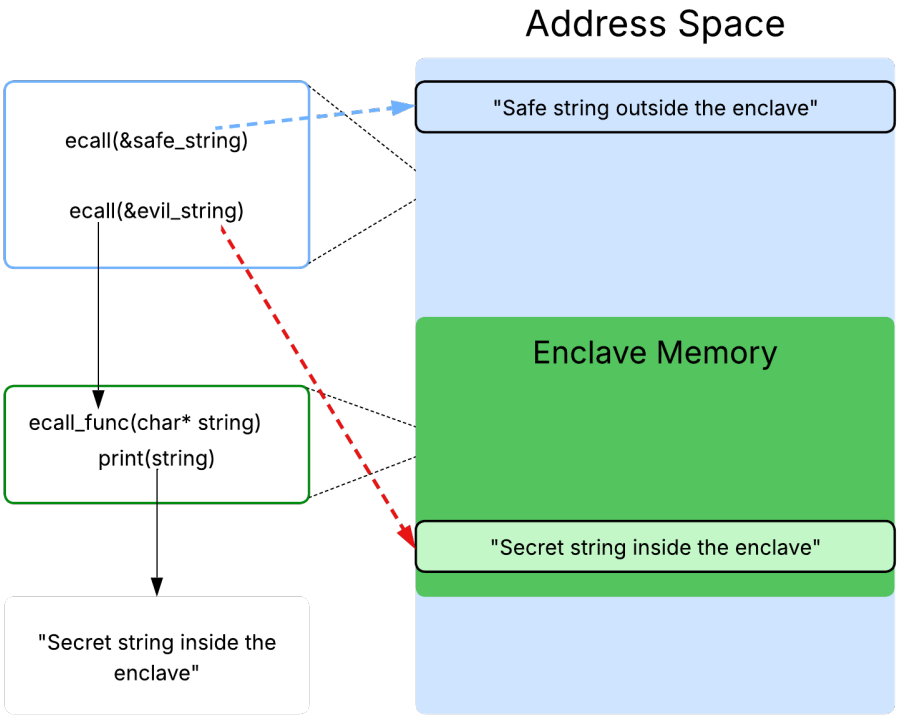


Figure 7.3: Conceptually illustrating what can go wrong if pointers are not validated. Here, the enclave code expects a pointer to a string outside its own memory, but is given a pointer to a secret string in its own memory, which is subsequently "printed".

vector that should been considered when using TEEs.

Responsibility and Engineering Culture

This also raises a governance question: *who is responsible for securing this ecosystem?* Intel provides the hardware and the initial SDKs, but has historically lacked a strong software security culture. Microsoft, by contrast, is actively developing the Open Enclave SDK[2] and has brought more engineering discipline to this space. Given this, it may be reasonable to prefer a shielding runtime developed by a software-focused company.

Intel's own guidance suggests that the Independent Software Vendor (ISV) should bear primary responsibility for securely partitioning applications, stating that "Since the ISV knows the application best, the ISV should conduct a security analysis of the application and properly partition it, making the decision about what code and data is placed in the enclave" [203]. However, in practice, developers working with complex third-party libraries or legacy code often lack the deep application-specific insight or resources to perform thorough security analysis. This can leave a gap in accountability: platform providers expect ISVs to partition securely, while ISVs may rely on SDKs or shielding runtimes to handle security. If each party assumes the other will ensure robust protection, oversights can easily arise, leading to vulnerabilities not from malicious intent but from misplaced expectations. A study [253] of production SGX systems found that debug enclaves were frequently deployed in the wild, disabling essential protections like sealing and attestation. These failures were often due to misconfiguration or unclear platform guidance.

Interestingly, despite the clear overlap in ABI responsibilities, each runtime currently maintains its own implementation. Van Bulck et al. argue for a unified ABI shielding layer [252], which I find persuasive. Such a unified effort could benefit from formal verification and better test coverage. However, it also raises concerns about monoculture: a bug in the unified layer could affect all users. Recent tools like Pandora [251] show promise in formally verifying shielding runtime behavior. Pandora systematically explores enclave entry and exit paths using symbolic execution, discovering bugs in major runtimes that had gone unnoticed. This approach, if effective enough, could provide a viable option for securing the tRTS and uRTS interactions.

Reflection

The software that enables interactions between the enclave (tRTS) and the rest of the enviornment (uRTS), is perhaps the most underestimated part of enclave application development. It is easy to focus on hardware capabilities or cryptographic properties, but the glue code (the SDKs, the ABI stubs, the runtime handlers) often determines the real-world security. As the TEE ecosystem matures, em-

[2] `https://github.com/openenclave/openenclave`

phasis must shift to these interfaces. Better abstractions, unification efforts, and formal verification are positive trends, but they do not yet eliminate the fundamental challenges of securing the trusted/untrusted boundary. In many ways, the future of TEE security lies not in silicon, but in software engineering discipline around these runtimes. Even if NSOs choose to outsource enclave development to an ISV or rely on shielding runtimes, they remain ultimately responsible for understanding and managing the risks associated with their data and threat model. Without informed oversight, NSOs risk assuming security guarantees they cannot verify or enforce, undermining the trust model of the entire system.

7.2.2 Development Platform

As discussed in Section 5.3, developing secure SGX applications requires navigating a diverse ecosystem of SDKs, shielding runtimes, and library OS frameworks. While these options offer flexibility, they also introduce complexity in choosing and maintaining an appropriate runtime environment. This diversity means NSOs must carefully assess the long-term security posture of third-party runtimes they rely on, as weaknesses in shielding runtimes or LibOS implementations can undermine even well-designed enclave applications.

A critical yet often underestimated aspect is management of signing keys. Because SGX attestation and sealing rely on cryptographic keys associated with enclave author, any compromise of these keys can enable attackers to forge enclaves that appear genuine. This creates a single point of failure in the development chain. Effective key protection (e.g., storing signing keys in HSMs or TPMs) is therefore not just a best practice but a necessity for any organization planning to distribute enclaves across systems they do not directly control.

Moreover, the development platform's own integrity is important: a compromised build environment can silently insert malicious code into enclaves before signing, invalidating all downstream trust. This risk is analogous to supply chain attacks seen in other domains, where attackers target build servers or developer workstations.

Overall, choosing a development platform is not only a matter of convenience or compatibility. It is a fundamental security decision shaping the trustworthiness of the entire TEE deployment.

7.2.3 Patching and Lifecycle Management

Once deployed, SGX enclaves are cryptographically sealed and signed, locking their code and measurements. Updating an enclave requires re-signing and potentially re-attestation, complicating automated patching workflows. This limitation creates friction for continuous integration and deployment processes, as every security fix or feature update must go through enclave rebuild, signing, and sometimes even hardware provisioning. In contrast, TEEs like AMD SEV, which operate at the virtual machine level, can rely on conventional OS and package updates without requiring enclave-specific re-deployment.

A fundamental challenge is that defending TEEs against new threats often becomes a security arms race: attackers continuously develop novel microarchitectural and software attacks, while vendors respond with patches and mitigations. However, these patches are typically spot mitigations —fixes that block known attack techniques without fundamentally eliminating the root cause. Over time, this pattern leads to an accumulation of mitigations layered on top of each other, increasing complexity, performance costs, and the risk of regression.

While TEE microcode updates theoretically allow hardware-level fixes, they are often incomplete or impractical to deploy. Microcode updates depend on BIOS or firmware upgrades, which may be missed in field systems, especially outside datacenter environments. Moreover, microcode cannot alter fundamental architectural design choices, limiting its ability to fully remediate certain classes of vulnerabilities.

A critical yet often overlooked component of secure enclave maintenance is the *TCB Recovery* process, which allows Intel to invalidate compromised TCB states and re-establish trust through microcode updates and attestation key refreshes. Recent SGX TCB recovery cycles for 4th and 5th Gen Xeon processors illustrate how Intel enforces updated microcode and BIOS versions to maintain attestation validity, preventing outdated systems from successfully attesting until they are patched [254]. While TCB recovery is a powerful mechanism for enforcing updated security postures, implementing it in practice is non-trivial: it requires platform owners to re-register affected hardware, update attestation services or PCCS caches, and rebuild enclaves with incremented ISVSVNs. These operational demands highlight that, although the technical facilities for TCB recovery exist, their effectiveness hinges on organizations' sustained commitment to timely updates and robust maintenance practices throughout the enclave lifecycle.

Analyses in the *sgx.fail* study demonstrate how these practical hurdles, combined with real-world constraints on enclave testing, signing processes, and large-scale deployment, can create significant security gaps that even dedicated vendors struggle to close [121]. These challenges show that TEE security is not guaranteed by technology alone but relies critically on organizations' ability to keep pace with an evolving threat landscape.

In effect, secure deployment of TEEs is not a one-time effort but an ongoing commitment. Organizations adopting TEEs like SGX must plan for continuous security monitoring, timely patching, and periodic reassessment of both enclave code and platform firmware. Without this, even the best-designed enclave architectures risk erosion of their security guarantees over time.

7.2.4 Long-Term Ecosystem Viability

Building on the feasibility observations in Section 5.3.5, it is important to critically evaluate the sustainability of SGX-based solutions for deployments requiring multi-decade support cycles, as is common in public-sector and government environments. Intel's discontinuation of SGX support on some recent desktop CPU

generations, combined with opaque dependency on proprietary architectural enclaves (e.g., the QE) and Intel's platform provisioning infrastructure, places developers at risk of losing functionality if vendor priorities change.

This tight coupling creates a classic vendor lock-in scenario: transitioning to an alternative TEE platform (such as AMD SEV-SNP or Arm's Confidential Compute solutions) typically requires substantial reengineering effort. As frameworks like OpenEnclave illustrate, cross-TEE abstractions remain limited in scope, and do not fully alleviate incompatibility in attestation models, memory management, or privilege levels. Even if source code is available, differences in hardware semantics can necessitate deep modifications, eroding portability.

A further concern is the maintenance status of critical SDKs and runtimes. At the time of writing, several notable SGX development frameworks, once prominent in the ecosystem, have seen little or no active maintenance. For example, the Apache Teaclave SGX SDK[3] has not had a commit in over two years, with its last official release dating back to 2019; R3's Conclave SDK[4], also shows no activity for two years; and Google's Asylo framework[5] framework, despite initial momentum, has been inactive for three years with its last release in 2021 (also discussed in Section 5.1). This stagnation highlights the risk that organizations may find themselves dependent on tools that lack ongoing security updates, compatibility fixes for new hardware, or timely responses to emerging vulnerabilities.

NSOs and other long-term stakeholders should therefore not only assess current technical capabilities but also Intel's track record, future platform roadmaps, and the health of the broader SGX software ecosystem. Factoring in potential end-of-life scenarios, the availability of active development communities, and contractually guaranteed support is essential to mitigate operational and security risks posed by ecosystem stagnation or unexpected platform discontinuations.

7.3 Trust and Information Flow

The word "trust" is central to this thesis, and to TEEs themselves. Concepts like the **Trusted** Execution Environment, **Trusted** Computing Base, and **Trusted** Runtime all rely on the notion that certain components must be assumed reliable. In this section, we examine what trust means in this context, how it differs from conventional use, and how it influences the design of a system such as the one proposed in Chapter 6.

[3]Teaclave SGX Github repository: `https://github.com/apache/incubator-teaclave-sgx-sdk`

[4]Conclave SDK Github repository: `https://github.com/R3Conclave/conclave-core-sdk`, inactive for over two years.

[5]Google Asylo Github repository: `https://github.com/google/asylo`, no commits in over three years, last release in 2021.

7.3.1 Trust and Trustworthiness

Adopting a hardware-based TEE, such as Intel SGX, reduces the trusted computing base but shifts significant responsibility onto the hardware vendor. This reflects a broader point: trust in the TCB is less a desirable property than a liability. If a trusted component fails or is compromised, the entire system can break. The system model's use of a custom attestation service (Section 6.2.1) reduces external dependencies, but trust in the vendor (Intel in case of SGX) remains unavoidable. This recursive dilemma is common in secure systems. Some root of trust must be assumed, even if it cannot be verified by the end user [62].

As discussed in the security analysis (Section 3.6), TEEs remain susceptible to side-channel and microarchitectural vulnerabilities. These risks show why even the most foundational trust anchors must be treated with caution. A technically "secure" system is not always a trustworthy one, it may protect against adversaries but still act against the interests of users or data owners. A

Ferguson et al. [231] stress that in cryptographic systems, trust is often contractual. That is, trust is not just technical but shaped by the roles and expectations between parties. In practice, TEE-based systems follow a similar pattern. Developers, data owners, CSPs, and hardware vendors must agree——explicitly or implicitly——on who controls what and under what assumptions. SGX's enclave signing process is a concrete example. Only enclaves signed with keys approved by Intel can produce attestations tied to the processor's hardware key. This preserves the platform's integrity but centralizes authority and limits sovereignty for organizations building on it. Even when attestation services are custom, meaningful verification still depends on Intel's infrastructure.

Trust is not only about cryptographic assurances. As Bursell emphasizes [62], trust relationships in computing depend on clarity and transparency. SGX provides partial transparency through attestation, but compartmentalized system designs reduce visibility into internal data flows. Data owners know where their data is initially sent, but may not be able to track how it is used thereafter. As a result, auditors often serve as the primary trust-enforcing mechanism, bridging the gap between opaque execution and verifiable policy compliance [62].

Still, audits alone are limited. Hardware and firmware complexity can obscure vulnerabilities, and full guarantees are out of reach without unrealistic verification coverage. Clear boundaries of responsibility between the NSO, data owners, CSP, and hardware vendors help clarify where trust is placed and how it can be evaluated. This becomes particularly important in systems handling sensitive statistical data, where the cost of misplaced trust is high.

To make these distinctions actionable, it is useful to separate what is trusted, what is secure, and what is trustworthy. A trusted component has the ability to break security, it must be assumed correct. A secure system withstands known attacks, but may not act in the user's interest. A trustworthy system is one that is secure and aligned with user goals. In that sense, trustworthiness is not a given; it must be earned. Strategies that move toward this goal include minimizing the

TCB, combining multiple independent roots of trust, using structured transparency to enforce policy, and introducing third-party audit mechanisms [255]. These are not silver bullets, but pragmatic ways to align the system's guarantees with the expectations of those who depend on it.

7.3.2 Information Flow and Control

The system model grants data owners control through structured transparency and policy-enforced attestation. Rather than relying solely on infrastructure-level privacy mechanisms, the model defines conditions under which data can be released, explicitly codified in attestation policies. This reflects ideas from Trask et al. [256], where structured transparency means that stakeholders agree on operational policies before any data exchange occurs.

This has practical implications. Remote and local attestation allow data owners to verify that enclaves meet agreed conditions before provisioning data. This level of control stands in contrast to cryptographic methods like MPC or DP, where enforcement is often implicit and tied to system design, not policy. Attestation allows verification of deployed code and system state, anchoring trust in something tangible and inspectable.

Compared to other privacy-preserving methods, this gives data owners more leverage. Instead of having to rely on reputational trust or legal guarantees, they can inspect, verify, and decide whether to participate based on concrete system properties. While it does not eliminate all risks, this model gives stakeholders a clear interface for making trust decisions, improving both transparency and accountability in practice.

7.4 Broader Context and Outlook

While the preceding sections have examined TEEs through the lens of hardware flaws, software complexity, and lifecycle burdens, these technologies also exist within a broader ecosystem of privacy-preserving technologies and rapidly evolving industry trends. For National Statistical Offices (NSOs), adopting TEEs is not solely a technical decision—it must be informed by the maturity of the surrounding ecosystem, the direction of platform vendor roadmaps, and the comparative strengths of alternative approaches such as DP, MPC, and .

This section offers perspective on how TEEs are positioned within the wider privacy landscape. It reflects on the maturity of current TEE implementations, industry trust dynamics, the historical challenges of isolation technologies, and how TEEs complement or compete with other privacy-preserving techniques. These broader considerations help determine whether TEEs are a short-term workaround or a long-term architectural choice for secure statistical computing in the cloud.

7.4.1 TEE Maturity and Industry Perspective

As highlighted by the attacks discussed in Section 3.6, Trusted Execution Environments (TEEs) have become a prime focus of modern systems security research. Many exploits require elevated (root) privileges to access model-specific registers (MSRs) or privileged instructions. While such capabilities imply full system compromise, TEEs are specifically designed to preserve security guarantees even in the presence of a malicious operating system. This makes them a uniquely attractive target: they promise isolation in precisely the environments where conventional protections fail.

This promise has elevated TEEs to a kind of stress test for processor security. Research of their security has "fueled" entire classes of microarchitectural vulnerabilities that extend beyond TEEs themselves [257]. Many of the insights that have shaped our understanding of speculative execution, buffer reuse, and transient leaks were first uncovered by attacking the assumptions that TEEs relied on. In this sense, TEEs have contributed to the hardening of platforms more broadly, even when the TEEs themselves remain imperfect.

Meanwhile, industry engagement has shown signs of improvement. Hardware vendors have begun working more transparently with the security community prior to product releases. For example, Intel and Microsoft jointly conducted a third-party security review of Intel TDX with both Microsoft [99] and Google [104][6], and AMD SEV-SNP has undergone external audits by teams like Google Project Zero [258]. The security researchers had access to documentation, source code, and not least the engineers responsible for designing and implementing the mechanisms. This auditing plays an important role in increasing trust in TEEs [255]. It marks a meaningful shift from the previous model of reactive patching toward more proactive and transparent assurance.

However, the path to robust security at these foundational levels remains challenging due to the scarcity of developers with expert knowledge in hardware, firmware, and OS intricacies, as visually illustrated in Figure 7.4. This highlights a potential disconnect between engineering needs and the industry's response. Rodrigo Branco, a former Chief Security Researcher at Intel, articulated this concern in his 2023 *hardwear.io* keynote, arguing that the industry's approach often prioritizes marketing or legal considerations over truly robust technical solutions [259]. His explicit caution regarding the emerging confidential computing ecosystem underscores this tension:

> "The rise of confidential computing will bring new reliability problems (and hide even further compromises)—a lot of technology has yet to be developed—anyone jumping into using it should not lie to themselves they are doing it for security reasons."

While current TEEs fall short of providing complete isolation, they remain one of the few tools available for securing computation in untrusted environments.

[6] Intel engineers are listed in the Acknowledgments.

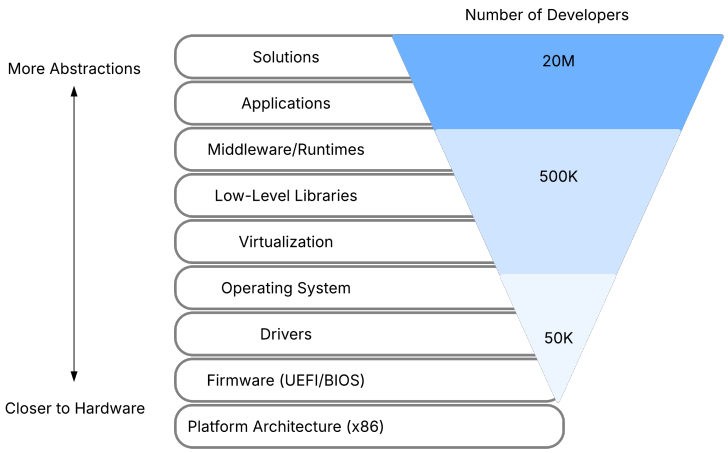


Figure 7.4: Illustrates the destiny of developers at each level of the software stack. Adapted from [260].

Their design has exposed hard problems at the boundary of hardware and software, problems that vendors are only beginning to confront more openly. Continued involvement from independent researchers has improved transparency and led to stronger designs, but many technical challenges remain unresolved.

7.4.2 Comparison with other PETs

While technical distinctions between TEEs and other PETs, such as HE, DP, and MPC, have been outlined in Section 2.3, the more relevant question for an NSO is how these approaches differ in terms of operational assumptions, deployment complexity, and policy control.

TEEs offer a hardware-based isolation model that, unlike cryptographic approaches, relies on enclave integrity and attestation rather than mathematical guarantees. This makes them comparatively flexible to deploy: a correctly attested enclave can execute general-purpose code without requiring the significant protocol engineering effort often associated with MPC, or the performance cost of HE. This makes TEEs particularly suited for situations where data must be processed across administrative boundaries but infrastructure is limited or heterogeneous.

One unique aspect of TEEs is their ability to enforce policy as part of the computation logic. Because enclaves can condition data access, usage, and output on runtime attestation results, they support a form of decentralized trust negotiation: each party can verify what code is running before releasing data, without requiring global coordination. TEEs can also produce audit logs tied to enclave identities, which is valuable in compliance-driven contexts such as national statistics.

At the same time, TEEs do not offer the same formal guarantees as DP or threshold-based MPC. Side-channel resistance and hardware trust remain open challenges. But for many NSO workflows, particularly those involving complex

record linkage or aggregation over sensitive but non-high-stakes data, TEEs represent a practical middle ground. They enable verifiable, policy-governed computation in cloud environments where full cryptographic solutions may be impractical.

Rather than replacing other PETs, TEEs complement them. For example, an enclave-based system can apply differential privacy as a final step, or use masked linkage schemes to reduce raw data exposure. In this light, TEEs are best seen not as a privacy silver bullet, but as a deployable enforcement mechanism that brings real-world feasibility to multi-party computation in untrusted infrastructure.

7.4.3 Current Trends and Research Directions in TEEs

As mentioned in Section 5.1, cloud providers are increasingly moving toward full-VM isolation with technologies like Intel TDX and AMD SEV-SNP (Section 3.4, Section 3.5), reducing reliance on enclave-based models such as SGX. This shift simplifies deployment and better fits existing cloud infrastructure, but does increase the TCB.

In parallel, there is growing interest in open-source components. Microsoft's OpenHCL [230], an open-source paravisor with support for TDX, aims to make the virtualization layer more auditable and is reportedly nearing production readiness. Marghera [261] takes a research approach to securing confidential VMs by preventing cross-VM leakage through stronger microarchitectural isolation.

Academic efforts are also exploring alternatives to vendor-controlled TEEs. Open and customization TEE platforms based on RISC-V such as Keystone [262], Dorami [263], and Penglai [264] enable experimentation with enclave models and hardware separation. Sancus [265] targets embedded systems without virtual memory, and POSTER [266] supports writing distributed applications across SGX, TrustZone, and Sancus.

A recurring motivation behind these projects is to reduce centralized trust and enable formal verification. Mühlberg and Van Bulck [267] argue that TEEs should be based on open designs, especially after the wave of microarchitectural vulnerabilities like Meltdown and Spectre. OpenTitan [268], an open[7] silicon RoT project, fits into this trend by making low-level trust anchors more transparent and auditable.

Whether these efforts converge remains to be seen, but they highlight that the future of TEEs is likely to include both industrial VM-level solutions and more modular, open alternatives.

7.4.4 Sustainability and Societal Impact

Though not the main focus of this thesis, the secure and privacy-preserving use of cloud infrastructures via TEEs supports several UN Sustainable Development Goals (SDGs) by enabling responsible data handling.

[7] Github Repository for OpenTitan: `https://github.com/lowRISC/opentitan`

From an **environmental perspective**, shifting NSO workloads to optimized cloud data centers can reduce energy consumption [269, 270] and electronic waste [271], aligning with SDG12 and SDG13 [272]. However, the additional overhead of TEEs and cryptographic protocols (see subsection 5.3.6) may offset some efficiency gains.

Regarding **social sustainability**, TEEs can contribute to SDG16 (peaceful and inclusive societies) by enabling confidentiality and integrity in processing sensitive data [273, 274], which is essential for public trust and safeguarding human rights [275]. This supports more secure, ethical statistical analysis, especially in sectors like healthcare (SDG3) and social equality (SDG10) [276, 277]. A recent example highlighting the positive societal potential of TEEs is the *Hope for Justice initiative* [278], which used SGX enclaves to securely match sensitive data on human trafficking victims across organizations, demonstrating how privacy-preserving computation can contribute to human rights efforts.

Critical reflection: Realizing these benefits requires managing energy sourcing, cryptographic efficiency, and governance to mitigate centralization risks inherent in cloud-based TEEs.

Chapter 8

Conclusion

In this thesis, we demonstrated that TEEs offer a viable and pragmatic path toward enabling privacy-preserving statistical computations, particularly for record linkage by NSOs, within otherwise untrusted public cloud environments. We addressed three key questions: the feasibility of designing secure TEE-based architectures for NSOs; the adequacy of commercial TEEs in meeting confidentiality and integrity requirements; and the real-world adoption challenges.

Answering the Research Questions

RQ1: *How can TEEs be architected into a system that enables NSOs to process sensitive data in the cloud with strong confidentiality and integrity guarantees?*

Using the proposed architecture we show that Intel SGX can protect sensitive identifiers during record linkage. Enclave-based attestation establishes trust, while hardware-enforced isolation defends against unauthorized access. However, the historical oversight of robust security practices within TEE software aspects, such as runtimes and LibOS, highlights the critical need for developers to explicitly define trusted code boundaries and apply rigorous interface hygiene to prevent vulnerabilities arising from software-level design flaws.

RQ2: *Do existing commercial TEEs provide security assurances sufficient to meet the privacy requirements of official statistics workflows?*

Commercial TEEs offer strong protection against many software-level threats, including a compromised OS or hypervisor. However, their reliance on complex microarchitectural designs introduces inherent challenges, making the complete elimination of side-channel and other microarchitectural attacks difficult. Consequently, while TEEs substantially improve confidentiality over standard cloud solutions, their guarantees fall short of absolute protection, particularly against these low-level hardware-based threats.

RQ3: *What practical and technical challenges arise when adopting TEEs for privacy-preserving record linkage, and do they render this approach feasible for real-world NSO deployments?*

TEE adoption is hindered by non-trivial development overhead, complex attestation processes, limited compatibility across vendor ecosystems, and the challenge of a nascent and often fragmented tooling ecosystem, where popular SDKs and runtimes may lack sustained maintenance. Despite this, they remain viable for NSO workflows when paired with institutional risk management, especially if implemented via shielding runtimes or confidential VMs for ease of deployment.

Key Findings

TEEs remain an evolving technology, with both architectural designs and surrounding tooling continuing to develop. We identified the following key observations:

- TEEs provide a practical foundation for secure statistics in the public cloud.
- Despite their advancements, fundamental design decisions in modern processor design pose inherent difficulties in fully eliminating side-channel and microarchitectural attacks.
- The security of TEE software components, including runtimes and LibOS layers, has historically been a source of vulnerabilities due to insufficient focus on their attack surface and interaction complexities.
- The TEE ecosystem presents practical challenges, including significant development overhead and the risk of relying on SDKs and runtimes that may not receive consistent long-term maintenance, impacting their viability for sustained real-world deployments.
- Design choices—such as adopting a LibOS versus manual partitioning—directly influence the trade-offs between security guarantees and implementation effort.
- While side-channel and microarchitectural attacks persist, TEEs demonstrably raise the baseline security posture compared to traditional cloud setups.

Limitations and Reflection

We present a concrete system architecture, but do not implement or evaluate the system in practice. Consequently, measurements regarding performance, scalability, or resistance to active attacks were not conducted. Furthermore, TEEs are evolving targets. The continued discovery of new vulnerabilities affecting prominent TEEs suggests that their security must be considered probabilistic, not absolute.

Nevertheless, the proposed design is thoroughly grounded in extensive research and rigorous analysis. This foundational work provides a realistic model for applying TEEs to privacy-preserving statistical computation and establishes a crucial framework for future empirical investigations.

8.1 Future Work

Future work should explore:

- Implementation and benchmarking of the proposed architecture under realistic loads.
- Combining TEEs with complementary techniques PETs, like MPC or DP to mitigate residual risks.
- Practical frameworks for enclave lifecycle management, attestation, and key provisioning at scale.
- Exploring the potential of open-source TEE implementations on open architectures like RISC-V, which could offer greater transparency, auditability, and potentially more straightforward mitigation strategies for microarchitectural vulnerabilities compared to proprietary designs.

Ultimately, TEEs do not guarantee perfect privacy, but they offer a tangible improvement over conventional cloud models. When treated as one component within a broader risk-aware architecture, they can enable secure and scalable statistical computing for public institutions. While the precise trajectory of confidential computing remains uncertain, current CSP offerings indicate a notable shift towards "next-generation" TEEs, such as confidential VMs like Intel TDX (see Section 3.4) and AMD SEV-SNP (see Section 3.5). These solutions prioritize ease of adoption, often at the cost of some security guarantees. Concurrently, academic efforts continue to explore new TEE designs for open architectures like RISC-V, which could provide a more transparent and auditable foundation, potentially easing the mitigation of complex microarchitectural vulnerabilities, hinting at further innovation in the field.

Bibliography

[1] United Nations Statistics Division, *Fundamental Principles of Official Statistics*, https://unstats.un.org/unsd/dnss/gp/FP-New-E.pdf, 2014.

[2] Eurostat, *European Statistics Code of Practice*, https://ec.europa.eu/eurostat/documents/4031688/8971242/KS-02-18-142-EN-N.pdf/e7f85f07-91db-4312-8118-f729c75878c7, Eurostat Statistical Working Paper, 2018.

[3] G. Spindler and P. Schmechel, 'Personal data and encryption in the european general data protection regulation,' *J. Intell. Prop. Info. Tech. & Elec. Com. L.*, vol. 7, p. 163, 2016.

[4] United Nations, *National Quality Assurance Frameworks Manual for Official Statistics: Including recommendations, the framework and implementation guidance*. New York: United Nations, 2019, ISBN: 9789210044769. DOI: 10.18356/1695ffd8-en. [Online]. Available: https://www.un-ilibrary.org/content/books/9789210044769.

[5] F. Ricciato, F. Ricciato, A. Bujnowska, A. Wirthmann, M. Hahn and E. Barredo-Capelot, 'A reflection on privacy and data confidentiality in official statistics,' Aug. 2019.

[6] G. Brackstone, 'Managing data quality in a statistical agency,' *Survey methodology*, vol. 25, no. 2, pp. 139–150, 1999.

[7] S. Signorelli, M. Fontana, L. Gabrielli and M. Vespe, 'Challenges and opportunities of computational social science for official statistics,' in *Handbook of Computational Social Science for Policy*, E. Bertoni, M. Fontana, L. Gabrielli, S. Signorelli and M. Vespe, Eds. Cham: Springer International Publishing, 2023, pp. 195–211, ISBN: 978-3-031-16624-2. DOI: 10.1007/978-3-031-16624-2_10. [Online]. Available: https://doi.org/10.1007/978-3-031-16624-2_10.

[8] F. Ricciato, A. Wirthmann and M. Hahn, 'Trusted smart statistics: How new data will change official statistics,' *Data & Policy*, vol. 2, e7, 2020. DOI: 10.1017/dap.2020.7.

[9] T. Koebe, 'Utilizing alternative data sources for official statistics,' Dissertation, Freie Universität Berlin, 2022. [Online]. Available: http://dx.doi.org/10.17169/refubium-36586.

[10] Cloud Security Alliance, *The Cloud Balancing Act for IT: Between Promise and Peril*, Survey Report, Press release (Jan. 13, 2015). Cloud Security Alliance, 2015. [Online]. Available: https://cloudsecurityalliance.org/press-releases/2016/01/13/csa-survey-64-9-of-it-trusts-the-cloud-as-much-or-more-than-on-premises-solutions/.

[11] R. Crowther, *More secure in the public cloud*, UK Defence Digital Official Blog, Head of Defence Digital Service explains why public cloud can be more secure than on-prem for government workloads, Nov. 2020. [Online]. Available: https://defencedigital.blog.gov.uk/2020/11/20/more-secure-in-the-public-cloud/.

[12] P. Thaine and G. Penn, 'Perfectly privacy-preserving AI what is it and how do we achieve it?' In *Proceedings of the PrivateNLP 2020: Workshop on Privacy in Natural Language Processing - Colocated with WSDM 2020, Houston, USA, Feb 7, 2020*, O. Feyisetan, S. Ghanavati, O. Rokhlenko and P. Thaine, Eds., ser. CEUR Workshop Proceedings, vol. 2573, CEUR-WS.org, 2020, pp. 37–38. [Online]. Available: https://ceur-ws.org/Vol-2573/PrivateNLP%5C_Poster2.pdf.

[13] P. Ohm, 'Broken promises of privacy: Responding to the surprising failure of anonymization,' *UCLA Law Review*, vol. 57, pp. 1701–1777, 2010, U of Colorado Law Legal Studies Research Paper No. 9-12. Available at SSRN: https://ssrn.com/abstract=1450006.

[14] D. W. Archer, B. de Balle Pigem, D. Bogdanov, M. Craddock, A. Gascon, R. Jansen, M. Jug, K. Laine, R. McLellan, O. Ohrimenko, M. Raykova, A. Trask and S. Wardley, *Un handbook on privacy-preserving computation techniques*, 2023. arXiv: 2301.06167 [cs.CY]. [Online]. Available: https://arxiv.org/abs/2301.06167.

[15] United Nations Global Working Group on Big Data for Official Statistics, *The united nations guide on privacy-enhancing technologies for official statistics*, https://unstats.un.org/bigdata/task-teams/privacy/, 2023.

[16] M. Schneider, R. J. Masti, S. Shinde, S. Capkun and R. Perez, *Sok: Hardware-supported trusted execution environments*, 2022. arXiv: 2205.12742 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2205.12742.

[17] M. Azure. 'Confidential computing: Elevating cloud security and privacy | confidential computing summit 2024.' (Dec. 2024), [Online]. Available: https://youtu.be/XssGI1q7Jak?si=TDZZU8jIWmD3pMCt&t=1607 (visited on 14/07/2025).

[18] Everest Group, 'Confidential computing – the next frontier in data security,' Everest Global, Inc., Tech. Rep., Oct. 2021, Licensed to the Confidential Computing Consortium.

[19] T. Harvey and R. Rodríguez, 'Hype cycle for compute, 2024,' Gartner, Inc., Tech. Rep., Jul. 2024, Confidential Computing is positioned in the Innovation Trigger phase. [Online]. Available: https://qant.com/gartner-hype-cycle-for-compute/.

[20] K. Grover, S. Tople, S. Shinde, R. Bhagwan and R. Ramjee, 'Privado: Practical and secure dnn inference with enclaves,' Sep. 2018. [Online]. Available: https://www.microsoft.com/en-us/research/publication/privado-practical-and-secure-dnn-inference-with-enclaves/.

[21] T. Lee, Z. Lin, S. Pushp, C. Li, Y. Liu, Y. Lee, F. Xu, C. Xu, L. Zhang and J. Song, 'Occlumency: Privacy-preserving remote deep-learning inference using sgx,' in *The 25th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '19, Los Cabos, Mexico: Association for Computing Machinery, 2019, ISBN: 9781450361699. DOI: 10.1145/3300061.3345447. [Online]. Available: https://doi.org/10.1145/3300061.3345447.

[22] K. Kim, C. H. Kim, J. " Rhee, X. Yu, H. Chen, D. (Tian and B. Lee, 'Vessels: Efficient and scalable deep learning prediction on trusted processors,' in *Proceedings of the 11th ACM Symposium on Cloud Computing*, ser. SoCC '20, Virtual Event, USA: Association for Computing Machinery, 2020, pp. 462–476, ISBN: 9781450381376. DOI: 10.1145/3419111.3421282. [Online]. Available: https://doi.org/10.1145/3419111.3421282.

[23] Y. Chen, F. Luo, T. Li, T. Xiang, Z. Liu and J. Li, 'A training-integrity privacy-preserving federated learning scheme with trusted execution environment,' *Information Sciences*, vol. 522, pp. 69–79, 2020, ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2020.02.037. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025520301201.

[24] F. Mo, H. Haddadi, K. Katevas, E. Marin, D. Perino and N. Kourtellis, 'Ppfl: Privacy-preserving federated learning with trusted execution environments,' in *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '21, Virtual Event, Wisconsin: Association for Computing Machinery, 2021, pp. 94–108, ISBN: 9781450384438. DOI: 10.1145/3458864.3466628. [Online]. Available: https://doi.org/10.1145/3458864.3466628.

[25] M. Birgersson, C. Artho and M. Balliu, 'Sharing without showing: Secure cloud analytics with trusted execution environments,' in *2024 IEEE Secure Development Conference (SecDev)*, 2024, pp. 105–116. DOI: 10.1109/SecDev61143.2024.00016.

[26] X. He, H. Wei, S. Han and D. Shen, 'Multi-party privacy-preserving record linkage method based on trusted execution environment,' in *Web Information Systems and Applications*, X. Zhao, S. Yang, X. Wang and J. Li, Eds.,

Cham: Springer International Publishing, 2022, pp. 591–602, ISBN: 978-3-031-20309-1.

[27] S. Han, K. Shen, D. Shen and C. Wang, 'Enhanced multi-party privacy-preserving record linkage using trusted execution environments,' *Mathematics*, vol. 12, no. 15, 2024, ISSN: 2227-7390. DOI: 10.3390/math12152337. [Online]. Available: https://www.mdpi.com/2227-7390/12/15/2337.

[28] P. Koeberl, V. Phegade, A. Rajan, T. Schneider, S. Schulz and M. Zhdanova, 'Time to rethink: Trust brokerage using trusted execution environments,' in *Trust and Trustworthy Computing*, M. Conti, M. Schunter and I. Askoxylakis, Eds., Cham: Springer International Publishing, 2015, pp. 181–190, ISBN: 978-3-319-22846-4.

[29] S. Warren and L. Brandeis, 'The right to privacy,' in *Killing the Messenger: 100 Years of Media Criticism*, Columbia University Press, 1989, pp. 1–21.

[30] D. Cole. "we kill people based on metadata'.' The New York Review. (May 2014), [Online]. Available: https://www.nybooks.com/online/2014/05/10/we-kill-people-based-metadata/.

[31] H. B. Newcombe, J. M. Kennedy, S. J. Axford and A. P. James, 'Automatic linkage of vital records,' *Science*, vol. 130, no. 3381, pp. 954–959, 1959. DOI: 10.1126/science.130.3381.954. eprint: https://www.science.org/doi/pdf/10.1126/science.130.3381.954. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.130.3381.954.

[32] I. P. Fellegi and A. B. S. and, 'A theory for record linkage,' *Journal of the American Statistical Association*, vol. 64, no. 328, pp. 1183–1210, 1969. DOI: 10.1080/01621459.1969.10501049. eprint: https://www.tandfonline.com/doi/pdf/10.1080/01621459.1969.10501049. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1080/01621459.1969.10501049.

[33] A. Sayers, Y. Ben-Shlomo, A. W. Blom and F. Steele, 'Probabilistic record linkage,' *International Journal of Epidemiology*, vol. 45, no. 3, pp. 954–964, Jun. 2016, Epub 2015 Dec 20. DOI: 10.1093/ije/dyv322. [Online]. Available: https://pmc.ncbi.nlm.nih.gov/articles/PMC5005943/.

[34] P. Christen, *Data Matching, Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection* (Data-Centric Systems and Applications). Springer Berlin, Heidelberg, 2012, ISBN: 978-3-642-31163-5. DOI: 10.1007/978-3-642-31164-2. [Online]. Available: https://link.springer.com/book/10.1007/978-3-642-31164-2.

[35] W. Guo, P. Pepitone, A. J. Aviv and M. L. Mazurek, 'How researchers de-identify data in practice,' in *Proceedings of the 34th USENIX Security Symposium (USENIX Security '25)*, To appear, Philadelphia, PA, USA: USENIX Association, Aug. 2025. [Online]. Available: https://www.usenix.org/s

ystem/files/conference/usenixsecurity25/sec25cycle1-prepub-631-guo-wentao.pdf.

[36] R. Hall and S. E. Fienberg, 'Privacy-preserving record linkage,' in *Privacy in Statistical Databases*, J. Domingo-Ferrer and E. Magkos, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 269–283, ISBN: 978-3-642-15838-4.

[37] A. Gkoulalas-Divanis, D. Vatsalan, D. Karapiperis and M. Kantarcioglu, 'Modern privacy-preserving record linkage techniques: An overview,' *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 4966–4987, 2021. DOI: 10.1109/TIFS.2021.3114026.

[38] M. Zhao E and Y. Geng, 'Homomorphic encryption technology for cloud computing,' *Procedia Computer Science*, vol. 154, pp. 73–83, 2019, Proceedings of the 9th International Conference of Information and Communication Technology [ICICT-2019] Nanning, Guangxi, China January 11-13, 2019, ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2019.06.012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050919307811.

[39] W.-j. Lu, S. Kawasaki and J. Sakuma, *Using fully homomorphic encryption for statistical analysis of categorical, ordinal and numerical data*, Cryptology ePrint Archive, Paper 2016/1163, 2016. DOI: http://dx.doi.org/10.14722/ndss.2017.23119. [Online]. Available: https://eprint.iacr.org/2016/1163.

[40] C. Gentry, 'A fully homomorphic encryption scheme,' crypto.stanford.edu/craig, Ph.D. dissertation, Stanford University, 2009.

[41] *Microsoft SEAL (release 4.1)*, https://github.com/Microsoft/SEAL, Microsoft Research, Redmond, WA., Jan. 2023.

[42] A. Viand, P. Jattke and A. Hithnawi, 'Sok: Fully homomorphic encryption compilers,' in *2021 IEEE Symposium on Security and Privacy (SP)*, 2021, pp. 1092–1108. DOI: 10.1109/SP40001.2021.00068. [Online]. Available: https://www.computer.org/csdl/proceedings-article/sp/2021/893400b166/1t0x938o5nW.

[43] A. Wood, M. Altman, A. Bembenek, M. Bun, M. Gaboardi, J. Honaker, K. Nissim, D. O'Brien, T. Steinke and S. Vadhan, 'Differential privacy: A primer for a non-technical audience,' *SSRN Electronic Journal*, Jan. 2018. DOI: 10.2139/ssrn.3338027.

[44] C. Dwork and A. Roth, 'The algorithmic foundations of differential privacy,' *Foundations and Trends® in Theoretical Computer Science*, vol. 9, no. 3–4, pp. 211–407, 2014, ISSN: 1551-305X. DOI: 10.1561/0400000042. [Online]. Available: http://dx.doi.org/10.1561/0400000042.

[45] S. Garfinkel, 'Differential Privacy and the 2020 US Census,' *MIT Case Studies in Social and Ethical Responsibilities of Computing*, no. Winter 2022, Jan. 2022, https://mit-serc.pubpub.org/pub/differential-privacy-2020-us-census.

[46] D. Desfontaines, *Why not differential privacy?* `https://desfontain.es/blog/why-not-differential-privacy.html`, Ted is writing things (personal blog), Mar. 2021.

[47] F. Tramer, A. Terzis, T. Steinke, S. Song, M. Jagielski and N. Carlini, *Debugging differential privacy: A case study for privacy auditing*, 2022. arXiv: `2202.12219 [cs.LG]`. [Online]. Available: `https://arxiv.org/abs/2202.12219`.

[48] D. Chaum, C. Crépeau and I. Damgard, 'Multiparty unconditionally secure protocols,' in *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, ser. STOC '88, Chicago, Illinois, USA: Association for Computing Machinery, 1988, pp. 11–19, ISBN: 0897912640. DOI: `10.1145/62212.62214`. [Online]. Available: `https://doi.org/10.1145/62212.62214`.

[49] D. Evans, V. Kolesnikov, M. Rosulek *et al.*, 'A pragmatic introduction to secure multi-party computation,' *Foundations and Trends® in Privacy and Security*, vol. 2, no. 2-3, pp. 70–246, 2018.

[50] Y. Lindell, *Secure multiparty computation (MPC)*, Cryptology ePrint Archive, Paper 2020/300, 2020. DOI: `10.1145/3387108`. [Online]. Available: `https://eprint.iacr.org/2020/300`.

[51] U. Maurer, 'Secure multi-party computation made simple,' *Discrete Applied Mathematics*, vol. 154, no. 2, pp. 370–381, 2006, Coding and Cryptography, ISSN: 0166-218X. DOI: `https://doi.org/10.1016/j.dam.2005.03.020`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0166218X05002428`.

[52] B. Knott, S. Venkataraman, A. Hannun, S. Sengupta, M. Ibrahim and L. van der Maaten, *Crypten: Secure multi-party computation meets machine learning*, 2022. arXiv: `2109.00984 [cs.LG]`. [Online]. Available: `https://arxiv.org/abs/2109.00984`.

[53] B. Broadnax, A. Koch, J. Mechler, T. Müller, J. Müller-Quade and M. Nagel, 'Fortified multi-party computation: Taking advantage of simple secure hardware modules,' *Proceedings on Privacy Enhancing Technologies*, 2021.

[54] L. Ren, Z. Liu, F. Li, K. Liang, Z. Li and B. Luo, 'Privdnn: A secure multiparty computation framework for deep learning using partial dnn encryption,' *Proceedings on Privacy Enhancing Technologies*, 2024.

[55] K. Hamada, D. Ikarashi, R. Kikuchi and K. Chida, *Efficient decision tree training with new data structure for secure multi-party computation*, 2021. arXiv: 2112.12906 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2112.12906.

[56] B. B. OBE, *World-leaders in cryptography: Phillip rogaway*, Cryptography Podcast, Oct. 2024. [Online]. Available: https://youtu.be/1ikrWaOYk5c?si=upLn3djYigQPPfLy&t=2326.

[57] J. Saltzer and M. Schroeder, 'The protection of information in computer systems,' *Proceedings of the IEEE*, vol. 63, no. 9, pp. 1278–1308, 1975. DOI: 10.1109/PROC.1975.9939.

[58] PaX Team, *Non-executable pages: Design and implementation*, https://pax.grsecurity.net/docs/noexec.txt, Accessed: 2025-07-02, 2003.

[59] M. Abadi, M. Budiu, Ú. Erlingsson and J. Ligatti, 'Control-flow integrity principles, implementations, and applications,' *ACM Trans. Inf. Syst. Secur.*, vol. 13, no. 1, Nov. 2009, ISSN: 1094-9224. DOI: 10.1145/1609956.1609960. [Online]. Available: https://doi.org/10.1145/1609956.1609960.

[60] B. W. Lampson, 'A note on the confinement problem,' *Commun. ACM*, vol. 16, no. 10, pp. 613–615, Oct. 1973, ISSN: 0001-0782. DOI: 10.1145/362375.362389. [Online]. Available: https://doi.org/10.1145/362375.362389.

[61] L. Szekeres, M. Payer, T. Wei and D. Song, 'Sok: Eternal war in memory,' in *2013 IEEE Symposium on Security and Privacy*, 2013, pp. 48–62. DOI: 10.1109/SP.2013.13.

[62] M. Bursell, *Trust in Computer Systems and the Cloud*. John Wiley & Sons, Ltd, 2020, ISBN: 9781119695158. DOI: 10.1002/9781119695158. [Online]. Available: https://onlinelibrary.wiley.com/doi/book/10.1002/9781119695158.

[63] Trusted Computing Group, *TCG TPM 2.0: Brief Overview*, https://trustedcomputinggroup.org/wp-content/uploads/2019_TCG_TPM2_BriefOverview_DR02web.pdf, Accessed: 2025-05-01, 2019.

[64] Google Cloud. 'Less trust, more security: The new model of cloud operations.' Accessed: 2025-05-15. (Feb. 2021), [Online]. Available: https://www.forbes.com/sites/googlecloud/2021/02/03/less-trust-more-security-the-new-model-of-cloud-operations/.

[65] Trusted Computing Group, *Tcg roots of trust specification, family "1.0", level 00, revision 0.20*, Draft specification for public review. Work in progress., Jul. 2018. [Online]. Available: https://trustedcomputinggroup.org/wp-content/uploads/TCG_Roots_of_Trust_Specification_v0p20_PUBLIC_REVIEW.pdf (visited on 15/05/2025).

[66] W. Arthur, D. Challener and K. Goldman, *A practical guide to TPM 2.0: Using the new trusted platform module in the new age of security*. Springer Nature, 2015.

[67] M. Sabt, M. Achemlal and A. Bouabdallah, 'Trusted execution environment: What it is, and what it is not,' in *Proceedings of the IEEE Trustcom/BigDataSE/ISPA*, 2015, pp. 57–64. DOI: 10.1109/Trustcom.2015.357.

[68] Intel Corporation, *Intel 64 and IA-32 Architectures Software Developer's Manual, volume 3: System programming guide*, Order Number 253668-077US, 2022.

[69] J. Rutkowska, 'Intel x86 considered harmful,' Invisible Things Lab, Tech. Rep., version 1.0, Oct. 2015. [Online]. Available: https://blog.invisiblethings.org/papers/2015/x86_harmful.pdf.

[70] A. Muñoz, R. Ríos, R. Román and J. López, 'A survey on the (in)security of trusted execution environments,' *Computers & Security*, vol. 129, p. 103 180, 2023, ISSN: 0167-4048. DOI: https://doi.org/10.1016/j.cose.2023.103180. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167404823000901.

[71] M. Li, Y. Yang, G. Chen, M. Yan and Y. Zhang, 'Sok: Understanding design choices and pitfalls of trusted execution environments,' in *Proceedings of the ACM Asia Conference on Computer and Communications Security (AsiaCCS)*, 2024.

[72] F. Zhang and H. Zhang, 'Sok: A study of using hardware-assisted isolated execution environments for security,' in *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016*, ser. HASP '16, Seoul, Republic of Korea: Association for Computing Machinery, 2016, ISBN: 9781450347693. DOI: 10.1145/2948618.2948621. [Online]. Available: https://doi.org/10.1145/2948618.2948621.

[73] M. Sommerhalder, 'Trusted execution environment,' in *Trends in Data Protection and Encryption Technologies*, V. Mulder *et al.*, Eds., Springer, 2023, pp. 95–101. DOI: 10.1007/978-3-031-33386-6_18. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-031-33386-6_18.

[74] W. Wang, L. Song, B. Mei, S. Liu, S. Ji, X. Wang, D. Yao and R. Hou, 'The road to trust: Building enclaves within confidential vms,' in *Network and Distributed System Security Symposium (NDSS)*, 2025. DOI: 10.14722/ndss.2025.240385.

[75] A. Baumann, M. Peinado and G. Hunt, 'Shielding applications from an untrusted cloud with haven,' *ACM Trans. Comput. Syst.*, vol. 33, no. 3, Aug. 2015, ISSN: 0734-2071. DOI: 10.1145/2799647. [Online]. Available: https://doi.org/10.1145/2799647.

[76] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue and U. R. Savagaonkar, 'Innovative instructions and software model for isolated execution.,' *Hasp@ isca*, vol. 10, no. 1, 2013.

[77] S. Gueron, 'A memory encryption engine suitable for general purpose processors,' *IACR Cryptol. ePrint Arch.*, vol. 2016, p. 204, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:5062796.

[78] S. P. Johnson, U. R. Savagaonkar, V. R. Scarlata, F. X. McKeen and C. V. Rozas, 'Technique for supporting multiple secure enclaves,' 8,972,746, US Patent, Dec. 2010.

[79] F. X. McKeen, C. V. Rozas, U. R. Savagaonkar, S. P. Johnson, V. Scarlata, M. A. Goldsmith, E. Brickell, J. T. Li, H. C. Herbert, P. Dewan *et al.*, 'Method and apparatus to provide secure application execution,' 9,087,200, US Patent, Dec. 2009.

[80] Intel Corporation, 'XuCode: An innovative technology for implementing complex instruction flows,' *Intel Developer Articles*, 2021, ID 758404, Updated 5/6/2021. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/secure-coding/xucode-implementing-complex-instruction-flows.html.

[81] V. Costan and S. Devadas, 'Intel sgx explained,' *IACR Cryptol. ePrint Arch.*, p. 86, 2016. [Online]. Available: https://eprint.iacr.org/2016/086.

[82] S. Brenner, C. Wulf, D. Goltzsche, N. Weichbrodt, M. Lorenz, C. Fetzer, P. Pietzuch and R. Kapitza, 'Securekeeper: Confidential zookeeper using intel sgx,' in *Proceedings of the 17th International Middleware Conference*, ser. Middleware '16, Trento, Italy: Association for Computing Machinery, 2016, ISBN: 9781450343008. DOI: 10.1145/2988336.2988350. [Online]. Available: https://doi.org/10.1145/2988336.2988350.

[83] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'Keeffe, M. L. Stillwell, D. Goltzsche, D. Eyers, R. Kapitza, P. Pietzuch and C. Fetzer, 'SCONE: Secure linux containers with intel SGX,' in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, Savannah, GA: USENIX Association, Nov. 2016, pp. 689–703, ISBN: 978-1-931971-33-1. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/arnautov.

[84] I. Corporation, *Technical overview of intel® sgx memory encryption architecture in xeon scalable processors*, https://www.intel.com/content/www/us/en/support/articles/000059614/processors/intel-xeon-processors.html, Accessed July 2025, 2024.

[85] Intel Corporation, *Intel® Software Guard Extensions (Intel® SGX) – Key Management on the 3rd Generation Intel® Xeon® Scalable Processor*, https://www.intel.com/content/www/us/en/content-details/706019/intel-software-guard-extensions-intel-sgx-key-management-on-the-3rd-generation-intel-xeon-scalable-processor.html, Document ID 706019, Intel IDZ Technical Library, Aug. 2021.

[86] Intel Corporation, 'Overview of an intel software guard extensions enclave life cycle,' Intel Corporation, Tech. Rep., 2016, Document ID 672746, updated 12/20/2016. [Online]. Available: https://www.intel.com/content/dam/develop/external/us/en/documents/intelsgxenclavelifecycle.pdf.

[87] S. Constable, J. V. Bulck, X. Cheng, Y. Xiao, C. Xing, I. Alexandrovich, T. Kim, F. Piessens, M. Vij and M. Silberstein, 'AEX-Notify: Thwarting precise Single-Stepping attacks through interrupt awareness for intel SGX enclaves,' in *32nd USENIX Security Symposium (USENIX Security 23)*, Anaheim, CA: USENIX Association, Aug. 2023, pp. 4051–4068, ISBN: 978-1-939133-37-3. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/constable.

[88] I. Anati, S. Gueron, S. Johnson and V. Scarlata, 'Innovative technology for cpu based attestation and sealing,' in *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy*, ACM New York, NY, USA, vol. 13, 2013.

[89] Intel Corporation, 'Introduction to intel® sgx sealing,' *Intel Developer Articles*, May 2016, ID 672637, Updated 5/4/2016. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/technical/introduction-to-intel-sgx-sealing.html.

[90] T. Knauth, M. Steiner, S. Chakrabarti, L. Lei, C. Xing and M. Vij, 'Integrating remote attestation with transport layer security,' *CoRR*, vol. abs/1801.05863, 2018. arXiv: 1801.05863. [Online]. Available: http://arxiv.org/abs/1801.05863.

[91] Y. Swami, 'Intel sgx remote attestation is not sufficient.(2017),' 2017.

[92] E. Brickell and J. Li, 'Enhanced privacy id from bilinear pairing for hardware authentication and attestation,' *International Journal of Information Privacy, Security and Integrity 2*, vol. 1, no. 1, pp. 3–33, 2011.

[93] S. Weiser and M. Werner, 'Sgxio: Generic trusted i/o path for intel sgx,' in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, ser. CODASPY '17, Scottsdale, Arizona, USA: Association for Computing Machinery, 2017, pp. 261–268, ISBN: 9781450345231. DOI: 10.1145/3029806.3029822. [Online]. Available: https://doi.org/10.1145/3029806.3029822.

[94] H. Liang, M. Li, Y. Chen, L. Jiang, Z. Xie and T. Yang, 'Establishing trusted i/o paths for sgx client systems with aurora,' *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 1589–1600, 2020. DOI: 10.1109/TIFS.2019.2945621.

[95] T. Peters, R. Lal, S. Varadarajan, P. Pappachan and D. Kotz, 'Bastion-sgx: Bluetooth and architectural support for trusted i/o on sgx,' in *Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy*, ser. HASP '18, Los Angeles, California: Association for Computing Machinery, 2018, ISBN: 9781450365000. DOI: 10.1145/3214292.3214295. [Online]. Available: https://doi.org/10.1145/3214292.3214295.

[96] M. Alharthi, F. Sang, D. Kuvaiskii, M. Vij and T. Kim, 'Rakis: Secure fast i/o primitives across trust boundaries on intel sgx,' in *Proceedings of the Twentieth European Conference on Computer Systems*, ser. EuroSys '25, Rotterdam, Netherlands: Association for Computing Machinery, 2025, pp. 1177–1193, ISBN: 9798400711961. DOI: 10.1145/3689031.3696090. [Online]. Available: https://doi.org/10.1145/3689031.3696090.

[97] R. Uhlig, G. Neiger, D. Rodgers, A. L. Santoni, F. C. Martins, A. V. Anderson, S. M. Bennett, A. Kagi, F. H. Leung and L. Smith, 'Intel virtualization technology,' *Computer*, vol. 38, no. 5, pp. 48–56, 2005.

[98] I. Corporation, 'Intel® TDX Module Base Architecture Specification,' Intel Corporation, Tech. Rep. 348549-005US, Oct. 2024, https://cdrdv2.intel.com/v1/dl/getContent/733575.

[99] M. Ben Hania, M. Villard, Y. Netzer, N. Kodalapura and T. Nguyen, 'Technical report of joint security review by microsoft and intel on intel tdx 1.5,' Microsoft Offensive Research & Security Engineering; Intel Offensive Security Research, Tech. Rep., Aug. 2024, Accessed 2025-07-04. [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/security-advisory/documents/intel_tdx_joint_security_review_with_microsoft.pdf.

[100] Intel Corporation, *Intel Trust Domain Extensions - SEAM Loader (SEAMLDR) Interface Specification*, urlhttps://cdrdv2.intel.com/v1/dl/getContent/739045, Reference Number 343755-003US, Mar. 2022.

[101] P.-C. Cheng, W. Ozga, E. Valdez, S. Ahmed, Z. Gu, H. Jamjoom, H. Franke and J. Bottomley, *Intel tdx demystified: A top-down approach*, 2023. arXiv: 2303.15540 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2303.15540.

[102] I. Corporation, 'Intel® Trust Domain Extensions,' Intel Corporation, Tech. Rep. 343961-003US, Feb. 2022.

[103] H. Khosravi, 'Runtime encryption of memory with intel® total memory encryption - multi-key,' Intel Corporation, Tech. Rep., Oct. 2022. [Online]. Available: https://www.intel.com/content/dam/www/central-libraries/us/en/documents/2022-10/intel-total-memory-encryption-multi-key-whitepaper.pdf.

[104] E. Aktas, C. Cohen, J. Eads, J. Forshaw and F. Wilhelm, 'Intel trust domain extensions (tdx) security review,' *Google security review*, 2023. [Online]. Available: https://services.google.com/fh/files/misc/intel_tdx_-_full_report_041423.pdf.

[105] I. Corporation, *Intel® TDX Connect Architecture Specification*, Overview and base architecture specification for Intel TDX Connect, 354629 001-US, Intel Corporation, Mar. 2023. [Online]. Available: https://cdrdv2.intel.com/v1/dl/getContent/773614.

[106] I. Corporation, 'Intel® TDX Connect: TEE-IO Device Guide,' Intel Corporation, Tech. Rep., Feb. 2023. [Online]. Available: https://cdrdv2-public.intel.com/772642/whitepaper-tee-io-device-guide-v0-6-5.pdf.

[107] I. Corporation, 'Software Enabling for Intel® TDX in Support of TEE-I/O,' Intel Corporation, Tech. Rep., version 1.00, Sep. 2022. [Online]. Available: https://cdrdv2-public.intel.com/742542/software-enabling-for-tdx-tee-io-fixed.pdf.

[108] I. Corporation, 'Intel® Trust Domain Extensions (Intel® TDX) Module: TD Partitioning Architecture Specification,' Intel Corporation, Tech. Rep. 354807-004US, Oct. 2024, Document ID: 839198. [Online]. Available: https://www.intel.com/content/www/us/en/content-details/839198/partitioning-architecture-specification-for-intel-tdx-v1-5.html.

[109] D. Kaplan, 'AMD x86 memory encryption technologies,' in *25th USENIX Security Symposium (USENIX Security 16)*, Austin, TX: USENIX Association, Aug. 2016. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/kaplan.

[110] D. Kaplan, J. Powell and T. Woller, *Amd memory encryption*, https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/white-papers/memory-encryption-white-paper.pdf, AMD White Paper, 2021.

[111] R. Buhren, C. Werling and J.-P. Seifert, 'Insecure until proven updated: Analyzing amd sev's remote attestation,' in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2019, pp. 1087–1099. DOI: 10.1145/3319535.3354207. [Online]. Available: https://arxiv.org/pdf/1908.11680.

[112] D. Kaplan, *Protecting vm register state with sev-es*, https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/white-papers/Protecting-VM-Register-State-with-SEV-ES.pdf, AMD White Paper, 2017.

[113] M. Li, Y. Zhang, H. Wang, K. Li and Y. Cheng, 'CIPHERLEAKS: Breaking constant-time cryptography on AMD SEV via the ciphertext side channel,' in *30th USENIX Security Symposium (USENIX Security 21)*, USENIX Association, Aug. 2021, pp. 717–732, ISBN: 978-1-939133-24-3. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/li-mengyuan.

[114] J. Werner, J. Mason, M. Antonakakis, M. Polychronakis and F. Monrose, 'The severest of them all: Inference attacks against secure virtual enclaves,' in *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, ser. Asia CCS '19, Auckland, New Zealand: Association for Computing Machinery, 2019, pp. 73–85, ISBN: 9781450367523. DOI: 10.1145/3321705.3329820. [Online]. Available: https://doi.org/10.1145/3321705.3329820.

[115] Advanced Micro Devices, *Amd sev-snp: Strengthening vm isolation with integrity protection and more*, https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/white-papers/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf, AMD White Paper, 2020.

[116] P. Paradzik, A. Derek and M. Horvat, 'Formal security analysis of the amd sev-snp software interface,' *arXiv preprint arXiv:2403.10296*, 2024. [Online]. Available: https://arxiv.org/pdf/2403.10296.

[117] B. Schlüter, S. Sridhara, A. Bertschi and S. Shinde, 'Wesee: Using malicious #vc interrupts to break amd sev-snp,' in *2024 IEEE Symposium on Security and Privacy (SP)*, 2024, pp. 4220–4238. DOI: 10.1109/SP54263.2024.00262.

[118] AMD, *SEV-SNP Platform Attestation Using VirTEE/SEV*, https://www.amd.com/content/dam/amd/en/documents/developer/58217-epyc-9004-ug-platform-attestation-using-virtee-snp.pdf, Publication, Issue Date: July, 2023; Revision 1.2 updated code to match library SEV-SNP update, Jul. 2023.

[119] Advanced Micro Devices, *Amd sev-tio: Trusted i/o for secure encrypted virtualization*, https://www.amd.com/content/dam/amd/en/documents/developer/sev-tio-whitepaper.pdf, AMD White Paper, 2023.

[120] Advanced Micro Devices, 'SEV-TIO Firmware Interface Specification,' Advanced Micro Devices, Technical Preview 58271, May 2023, Issue Date: May, 2023.

[121] S. Van Schaik, A. Seto, T. Yurek, A. Batori, B. AlBassam, D. Genkin, A. Miller, E. Ronen, Y. Yarom and C. Garman, 'Sok: Sgx.fail: How stuff gets exposed,' in *2024 IEEE Symposium on Security and Privacy (SP)*, 2024, pp. 4143–4162. DOI: 10.1109/SP54263.2024.00260.

[122] Y. Xu, W. Cui and M. Peinado, 'Controlled-channel attacks: Deterministic side channels for untrusted operating systems,' in *2015 IEEE Symposium on Security and Privacy*, 2015, pp. 640–656. DOI: 10.1109/SP.2015.45.

[123] M. Hähnel, W. Cui and M. Peinado, 'High-Resolution side channels for untrusted operating systems,' in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, Santa Clara, CA: USENIX Association, Jul. 2017, pp. 299–312, ISBN: 978-1-931971-38-6. [Online]. Available: https://www.usenix.org/conference/atc17/technical-sessions/presentation/hahnel.

[124] J. Van Bulck, F. Piessens and R. Strackx, 'SGX-Step: A practical attack framework for precise enclave execution control,' in *2nd Workshop on System Software for Trusted Execution (SysTEX)*, ACM, Oct. 2017, 4:1–4:6.

[125] J. Van Bulck and F. Piessens, 'SGX-Step: An open-source framework for precise dissection and practical exploitation of Intel SGX enclaves,' in *ACSAC 2023 Cybersecurity Artifacts Competition and Impact Award Finalist Short Paper*, Dec. 2023.

[126] L. Wilke, J. Wichelmann, A. Rabich and T. Eisenbarth, 'Sev-step a single-stepping framework for amd-sev,' *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2024, no. 1, pp. 180–206, 2024. DOI: 10.46586/TCHES.V2024.I1.180-206. [Online]. Available: https://doi.org/10.46586/tches.v2024.i1.180-206.

[127] L. Wilke, F. Sieck and T. Eisenbarth, 'Tdxdown: Single-stepping and instruction counting attacks against intel tdx,' in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '24, Salt Lake City, UT, USA: Association for Computing Machinery, 2024, pp. 79–93, ISBN: 9798400706363. DOI: 10.1145/3658644.3690230. [Online]. Available: https://doi.org/10.1145/3658644.3690230.

[128] Y. Yarom and K. Falkner, 'Flush+reload: A high resolution, low noise, l3 cache side-channel attack,' in *Proceedings of the 23rd USENIX Conference on Security Symposium*, ser. SEC'14, San Diego, CA: USENIX Association, 2014, pp. 719–732, ISBN: 9781931971157.

[129] D. Gruss, R. Spreitzer and S. Mangard, 'Cache template attacks: Automating attacks on inclusive last-level caches,' in *Proceedings of the 24th USENIX Conference on Security Symposium*, ser. SEC'15, Washington, D.C.: USENIX Association, 2015, pp. 897–912, ISBN: 9781931971232.

[130] B. Gras, K. Razavi, E. Bosman, H. Bos and C. Giuffrida, 'ASLR on the line: Practical cache attacks on the MMU,' in *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*, The Internet Society, 2017. [Online]. Available: https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/aslrcache-practical-cache-attacks-mmu/.

[131] C. Percival, *Cache missing for fun and profit*, 2005.

[132] D. A. Osvik, A. Shamir and E. Tromer, 'Cache attacks and countermeasures: The case of aes,' in *Proceedings of the 2006 The Cryptographers' Track at the RSA Conference on Topics in Cryptology*, ser. CT-RSA'06, San Jose, CA: Springer-Verlag, 2006, pp. 1–20, ISBN: 3540310339. DOI: 10.1007/11605805_1. [Online]. Available: https://doi.org/10.1007/11605805_1.

[133] A. Moghimi, G. Irazoqui and T. Eisenbarth, *CacheZoom: How SGX amplifies the power of cache attacks*, Cryptology ePrint Archive, Paper 2017/618, 2017. [Online]. Available: https://eprint.iacr.org/2017/618.

[134] F. Liu, Y. Yarom, Q. Ge, G. Heiser and R. B. Lee, 'Last-level cache side-channel attacks are practical,' in *2015 IEEE Symposium on Security and Privacy*, 2015, pp. 605–622. DOI: 10.1109/SP.2015.43.

[135] F. Brasser, U. Müller, A. Dmitrienko, K. Kostiainen, S. Capkun and A.-R. Sadeghi, 'Software grand exposure: SGX cache attacks are practical,' in *11th USENIX Workshop on Offensive Technologies (WOOT 17)*, Vancouver, BC: USENIX Association, Aug. 2017. [Online]. Available: https://www.usenix.org/conference/woot17/workshop-program/presentation/brasser.

[136] S. van Schaik, M. Minkin, A. Kwong, D. Genkin and Y. Yarom, 'Cacheout: Leaking data on intel cpus via cache evictions,' in *2021 IEEE Symposium on Security and Privacy (SP)*, 2021, pp. 339–354. DOI: 10.1109/SP40001.2021.00064.

[137] S. van Schaik, A. Kwong, D. Genkin and Y. Yarom, *SGAxe: How SGX fails in practice*, https://sgaxeattack.com/, 2020.

[138] M. Werner, T. Unterluggauer, L. Giner, M. Schwarz, D. Gruss and S. Mangard, 'Scattercache: Thwarting cache attacks via cache set randomization,' ser. SEC'19, Santa Clara, CA, USA: USENIX Association, 2019, pp. 675–692, ISBN: 9781939133069.

[139] B. Morgan, G. Horowitz, S. O'Connell, S. van Schaik, C. Chuengsatiansup, D. Genkin, O. Maennel, P. Montague, E. Ronen and Y. Yarom, *Slice+slice baby: Generating last-level cache eviction sets in the blink of an eye*, 2025. arXiv: 2504.11208 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2504.11208.

[140] Y. Yuan, Z. Liu, S. Deng, Y. Chen, S. Wang, Y. Zhang and Z. Su, ' CipherSteal: Stealing Input Data from TEE-Shielded Neural Networks with Ciphertext Side Channels,' in *2025 IEEE Symposium on Security and Privacy (SP)*, Los Alamitos, CA, USA: IEEE Computer Society, May 2025, pp. 4136–4154. DOI: 10.1109/SP61157.2025.00079. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP61157.2025.00079.

[141] S. Weiser, R. Spreitzer and L. Bodner, 'Single trace attack against rsa key generation in intel sgx ssl,' in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, ser. ASIACCS '18, Incheon, Republic of Korea: Association for Computing Machinery, 2018, pp. 575–586, ISBN: 9781450355766. DOI: 10.1145/3196494.3196524. [Online]. Available: https://doi.org/10.1145/3196494.3196524.

[142] M. Lipp, A. Kogler, D. Oswald, M. Schwarz, C. Easdon, C. Canella and D. Gruss, 'PLATYPUS: Software-based Power Side-Channel Attacks on x86,' in *2021 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2021.

[143] A. Kogler, J. Juffinger, L. Giner, L. Gerlach, M. Schwarzl, M. Schwarz, D. Gruss and S. Mangard, 'Collide+Power: Leaking inaccessible data with software-based power side channels,' in *32nd USENIX Security Symposium (USENIX Security 23)*, Anaheim, CA: USENIX Association, Aug. 2023, pp. 7285–7302, ISBN: 978-1-939133-37-3. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/kogler.

[144] G. Le Gonidec, G. Bouffard, J.-C. Prevotet and M. Méndez Real, 'Do not trust power management: A survey on internal energy-based attacks circumventing trusted execution environments security properties,' *ACM Trans. Embed. Comput. Syst.*, May 2025, Just Accepted, ISSN: 1539-9087. DOI: 10.1145/3735556. [Online]. Available: https://doi.org/10.1145/3735556.

[145] C. Canella, J. V. Bulck, M. Schwarz, M. Lipp, B. von Berg, P. Ortner, F. Piessens, D. Evtyushkin and D. Gruss, 'A systematic evaluation of transient execution attacks and defenses,' in *28th USENIX Security Symposium (USENIX Security 19)*, Santa Clara, CA: USENIX Association, Aug. 2019, pp. 249–266, ISBN: 978-1-939133-06-9. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/canella.

[146] C. Canella, K. N. Khasawneh and D. Gruss, 'The evolution of transient-execution attacks,' in *Proceedings of the 2020 on Great Lakes Symposium on VLSI*, ser. GLSVLSI '20, Virtual Event, China: Association for Computing Machinery, 2020, pp. 163–168, ISBN: 9781450379441. DOI: 10.1145/3386263.3407583. [Online]. Available: https://doi.org/10.1145/3386263.3407583.

[147] A. Fogh, *Covert shotgun*, https://cyber.wtf/2016/09/27/covert-shotgun/, Blog post, September 27, 2016. Automatically finding SMT covert channels, 2016.

[148] A. C. Aldaya, B. B. Brumley, S. ul Hassan, C. P. García and N. Tuveri, *Port contention for fun and profit*, Cryptology ePrint Archive, Paper 2018/1060, 2018. [Online]. Available: https://eprint.iacr.org/2018/1060.

[149] A. Bhattacharyya, A. Sandulescu, M. Neugschwandtner, A. Sorniotti, B. Falsafi, M. Payer and A. Kurmus, 'Smotherspectre: Exploiting speculative execution through port contention,' in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19, ACM, Nov. 2019, pp. 785–800. DOI: 10.1145/3319535.3363194. [Online]. Available: http://dx.doi.org/10.1145/3319535.3363194.

[150] J. Haj-Yahya, J. S. Kim, A. G. Yağlikçi, I. Puddu, L. Orosa, J. G. Luna, M. Alser and O. Mutlu, 'Ichannels: Exploiting current management mechanisms to create covert channels in modern processors,' in *Proceedings of the 48th Annual International Symposium on Computer Architecture*, ser. ISCA '21, Virtual Event, Spain: IEEE Press, 2021, pp. 985–998, ISBN: 9781450390866. DOI: 10.1109/ISCA52012.2021.00081. [Online]. Available: https://doi.org/10.1109/ISCA52012.2021.00081.

[151] Intel Corporation, *Intel® 64 and IA-32 Architectures Optimization Reference Manual, Volume 1*, 248966-050US, Manual, version 50, Section 3.4.1: Branch Prediction Optimization, Apr. 2024. [Online]. Available: https://cdrdv2.intel.com/v1/dl/getContent/671488.

[152] A. Fog, *The microarchitecture of intel, amd, and VIA cpus: An optimization guide for assembly programmers and compiler makers*, Last updated 2024-05-28, 2024. [Online]. Available: https://www.agner.org/optimize/microarchitecture.pdf.

[153] D. Evtyushkin, R. Riley, N. C. Abu-Ghazaleh, ECE and D. Ponomarev, 'Branchscope: A new side-channel attack on directional branch predictor,' in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '18, Williamsburg, VA, USA: Association for Computing Machinery, 2018, pp. 693–707, ISBN: 9781450349116. DOI: 10.1145/3173162.3173204. [Online]. Available: https://doi.org/10.1145/3173162.3173204.

[154] E. M. Koruyeh, K. N. Khasawneh, C. Song and N. Abu-Ghazaleh, 'Spectre returns! speculation attacks using the return stack buffer,' in *12th USENIX Workshop on Offensive Technologies (WOOT 18)*, Baltimore, MD: USENIX Association, Aug. 2018. [Online]. Available: https://www.usenix.org/conference/woot18/presentation/koruyeh.

[155] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin and T. H. Lai, 'Sgxpectre: Stealing intel secrets from sgx enclaves via speculative execution,' in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2019, pp. 142–157. DOI: 10.1109/EuroSP.2019.00020.

[156] M. H. I. Chowdhuryy, Z. Zhang and F. Yao, 'Powspectre: Powering up speculation attacks with tsx-based replay,' in *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, ser. ASIA CCS '24, Singapore, Singapore: Association for Computing Machinery, 2024, pp. 498–511, ISBN: 9798400704826. DOI: 10.1145/3634737.3661139. [Online]. Available: https://doi.org/10.1145/3634737.3661139.

[157] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom and M. Hamburg, 'Meltdown: Reading kernel memory from user space,' in *27th USENIX Security Symposium (USENIX Security 18)*, Baltimore, MD: USENIX Association, Aug. 2018, pp. 973–990, ISBN: 978-1-939133-04-5. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/lipp.

[158] J. V. Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom and R. Strackx, 'Foreshadow: Extracting the keys to the intel SGX kingdom with transient Out-of-Order execution,' in *27th USENIX Security Symposium (USENIX Security 18)*, Baltimore, MD: USENIX Association, Aug. 2018, 991–1008, ISBN: 978-1-939133-04-5. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/bulck.

[159] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom and R. Strackx, 'Breaking virtual memory protection and the sgx ecosystem with foreshadow,' *IEEE Micro*, vol. 39, no. 3, pp. 66–74, May 2019, ISSN: 0272-1732. DOI: 10.1109/MM.2019.2910104. [Online]. Available: https://doi.org/10.1109/MM.2019.2910104.

[160] S. van Schaik, A. Milburn, S. Österlund, P. Frigo, G. Maisuradze, K. Razavi, H. Bos and C. Giuffrida, 'RIDL: Rogue in-flight data load,' in *S&P*, May 2019.

[161] M. Schwarz, M. Lipp, D. Moghimi, J. Van Bulck, J. Stecklina, T. Prescher and D. Gruss, 'ZombieLoad: Cross-privilege-boundary data sampling,' in *CCS*, 2019.

[162] C. Canella, D. Genkin, L. Giner, D. Gruss, M. Lipp, M. Minkin, D. Moghimi, F. Piessens, M. Schwarz, B. Sunar, J. Van Bulck and Y. Yarom, 'Fallout: Leaking data on meltdown-resistant cpus,' in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, ACM, 2019.

[163] H. Ragab, A. Milburn, K. Razavi, H. Bos and C. Giuffrida, 'Crosstalk: Speculative data leaks across cores are real,' in *2021 IEEE Symposium on Security and Privacy (SP)*, 2021, pp. 1852–1867. DOI: 10.1109/SP40001.2021.00020.

[164] J. Van Bulck, D. Moghimi, M. Schwarz, M. Lipp, M. Minkin, D. Genkin, Y. Yuval, B. Sunar, D. Gruss and F. Piessens, 'LVI: Hijacking Transient Execution through Microarchitectural Load Value Injection,' in *41th IEEE Symposium on Security and Privacy (S&P'20)*, 2020.

[165] J. W. Sandro Rüegge and K. Razavi, 'Branch Privilege Injection: Compromising spectre v2 hardware mitigations by exploiting branch predictor race conditions,' in *34th USENIX Security Symposium (USENIX Security 25)*, 2025.

[166] K. Murdock, D. Oswald, F. D. Garcia, J. Van Bulck, D. Gruss and F. Piessens, 'Plundervolt: Software-based fault injection attacks against Intel SGX,' in *41st IEEE Symposium on Security and Privacy (S&P)*, May 2020, pp. 1466–1482.

[167] Z. Kenjar, T. Frassetto, D. Gens, M. Franz and A.-R. Sadeghi, 'V0ltpwn: Attacking x86 processor integrity from software,' in *Proceedings of the 29th USENIX Conference on Security Symposium*, ser. SEC'20, USA: USENIX Association, 2020, ISBN: 978-1-939133-17-5.

[168] Z. Chen, G. Vasilakis, K. Murdock, E. Dean, D. Oswald and F. D. Garcia, 'VoltPillager: Hardware-based fault injection attacks against intel SGX enclaves using the SVID voltage scaling interface,' in *30th USENIX Security Symposium (USENIX Security 21)*, USENIX Association, Aug. 2021, pp. 699–716, ISBN: 978-1-939133-24-3. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/chen-zitai.

[169] P. Qiu, D. Wang, Y. Lyu and G. Qu, 'Voltjockey: Breaching trustzone by software-controlled voltage manipulation over multi-core frequencies,' in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19, London, United Kingdom: Association for Computing Machinery, 2019, pp. 195–209, ISBN: 9781450367479. DOI: 10.1145/3319535.3354201. [Online]. Available: https://doi.org/10.1145/3319535.3354201.

[170] A. Tang, S. Sethumadhavan and S. Stolfo, 'CLKSCREW: Exposing the perils of Security-Oblivious energy management,' in *26th USENIX Security Symposium (USENIX Security 17)*, Vancouver, BC: USENIX Association, Aug. 2017, pp. 1057–1074, ISBN: 978-1-931971-40-9. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/tang.

[171] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai and O. Mutlu, 'Flipping bits in memory without accessing them: An experimental study of dram disturbance errors,' in *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ser. ISCA '14, Minneapolis, Minnesota, USA: IEEE Press, 2014, pp. 361–372, ISBN: 9781479943944.

[172] O. Mutlu and J. S. Kim, 'Rowhammer: A retrospective,' *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, vol. 39, no. 8, pp. 1555–1571, Aug. 2020, ISSN: 0278-0070. DOI: 10.1109/TCAD.2019.2915318. [Online]. Available: https://doi.org/10.1109/TCAD.2019.2915318.

[173] Y. Jang, J. Lee, S. Lee and T. Kim, 'Sgx-bomb: Locking down the processor via rowhammer attack,' in *Proceedings of the 2nd Workshop on System Software for Trusted Execution*, ser. SysTEX'17, Shanghai, China: Association for Computing Machinery, 2017, ISBN: 9781450350976. DOI: 10.1145/3152701.3152709. [Online]. Available: https://doi.org/10.1145/3152701.3152709.

[174] M. Lipp, M. Schwarz, L. Raab, L. Lamster, M. T. Aga, C. Maurice and D. Gruss, 'Nethammer: Inducing rowhammer faults through network requests,' in *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, 2020, pp. 710–719. DOI: 10.1109/EuroSPW51379.2020.00102.

[175] D. Gruss, M. Lipp, M. Schwarz, D. Genkin, J. Juffinger, S. O'Connell, W. Schoechl and Y. Yarom, 'Another flip in the wall of rowhammer defenses,' in *2018 IEEE Symposium on Security and Privacy (SP)*, 2018, pp. 245–261. DOI: 10.1109/SP.2018.00031.

[176] J. Cui, J. Z. Yu, S. Shinde, P. Saxena and Z. Cai, 'Smashex: Smashing sgx enclaves using exceptions,' in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '21, Virtual Event, Republic of Korea: Association for Computing Machinery, 2021, pp. 779–793, ISBN: 9781450384544. DOI: 10.1145/3460120.3484821. [Online]. Available: https://doi.org/10.1145/3460120.3484821.

[177] N. Weichbrodt, A. Kurmus, P. Pietzuch and R. Kapitza, 'Asyncshock: Exploiting synchronisation bugs in intel sgx enclaves,' in *Computer Security – ESORICS 2016*, I. Askoxylakis, S. Ioannidis, S. Katsikas and C. Meadows, Eds., Cham: Springer International Publishing, 2016, pp. 440–457, ISBN: 978-3-319-45744-4.

[178] B. Schlüter, S. Sridhara, M. Kuhne, A. Bertschi and S. Shinde, *Heckler: Breaking confidential vms with malicious interrupts*, 2024. arXiv: 2404.03387 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2404.03387.

[179] S. Sridhara, A. Bertschi, B. Schlüter and S. Shinde, *Sigy: Breaking intel sgx enclaves with malicious exceptions & signals*, 2024. arXiv: 2404.13998 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2404.13998.

[180] D. Moghimi, J. Van Bulck, N. Heninger, F. Piessens and B. Sunar, 'CopyCat: Controlled instruction-level attacks on enclaves,' in *29th USENIX Security Symposium*, Aug. 2020, pp. 469–486.

[181] F. Alder, J. Van Bulck, D. Oswald and F. Piessens, 'Faulty point unit: ABI poisoning attacks on Intel SGX,' in *36th Annual Computer Security Applications Conference (ACSAC)*, Dec. 2020, pp. 415–427.

[182] F. Alder, J. Van Bulck, J. Spielman, D. Oswald and F. Piessens, 'Faulty point unit: Abi poisoning attacks on trusted execution environments,' *Digital Threats: Research and Practice*, vol. 3, no. 2, pp. 1–26, Feb. 2022.

[183] P. Borrello, A. Kogler, M. Schwarzl, M. Lipp, D. Gruss and M. Schwarz, 'ÆPIC leak: Architecturally leaking uninitialized data from the microarchitecture,' in *31st USENIX Security Symposium (USENIX Security 22)*, Boston, MA: USENIX Association, Aug. 2022, pp. 3917–3934, ISBN: 978-1-939133-31-1. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/borrello.

[184] D. Lee, D. Jung, I. T. Fang, C.-C. Tsai and R. A. Popa, 'An off-chip attack on hardware enclaves via the memory bus,' in *Proceedings of the 29th USENIX Conference on Security Symposium*, ser. SEC'20, USA: USENIX Association, 2020, ISBN: 978-1-939133-17-5.

[185] J. De Meulemeester, L. Wilke, D. Oswald, T. Eisenbarth, I. Verbauwhede and J. Van Bulck, 'BadRAM: Practical memory aliasing attacks on trusted execution environments,' in *46th IEEE Symposium on Security and Privacy (S&P)*, May 2025.

[186] B. Kitchenham and S. Charters, 'Guidelines for performing systematic literature reviews in software engineering,' EBSE Technical Report EBSE-2007-01, Keele University, Tech. Rep., 2007.

[187] A. Carrera-Rivera, W. Ochoa, F. Larrinaga and G. Lasa, 'How-to conduct a systematic literature review: A quick guide for computer science research,' *MethodsX*, vol. 9, p. 101 895, 2022. DOI: 10.1016/j.mex.2022.101895.

[188] National Cyber Security Centre, *Protocol design principles*, https://www.ncsc.gov.uk/whitepaper/protocol-design-principles, 2020.

[189] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz and M. Russinovich, 'Vc3: Trustworthy data analytics in the cloud using sgx,' in *IEEE Symposium on Security and Privacy (S&P)*, 2015, pp. 38–54. DOI: 10.1109/SP.2015.10.

[190] M. Corporation, *Building and executing tee-based applications on azure*, version 1.2, White paper, Apr. 2020. [Online]. Available: https://download.microsoft.com/download/e/0/3/e03ff018-35ac-4bac-9638-3e45787bc23c/Building-and-Executing-TEE-based-applications-on-Azure-(April-2020).pdf.

[191] M. Corporation, *Leveraging attestations with tee-based applications on azure*, White paper, Jul. 2020. [Online]. Available: https://download.microsoft.com/download/e/0/3/e03ff018-35ac-4bac-9638-3e45787bc23c/Leveraging-Attestations-with-TEE-based-applications-on-Azure-(July-2020).pdf.

[192] M. Corporation, *Azure managed confidential consortium framework preview*, Blog post, 2022. [Online]. Available: https://techcommunity.microsoft.com/blog/azureconfidentialcomputingblog/microsoft-introduces-preview-of-azure-managed-confidential-consortium-framework/3648986.

[193] J. Young and S. Lugani. 'Announcing new confidential computing updates for even more hardware security options.' Accessed 2025-06-15. (Oct. 2024), [Online]. Available: https://cloud.google.com/blog/products/identity-security/new-confidential-computing-updates-for-more-hardware-security-options.

[194] J. Young. 'Announcing general availability of confidential gke nodes.' Accessed 2025-06-15. (Jun. 2022), [Online]. Available: https://cloud.google.com/blog/products/identity-security/announcing-general-availability-of-confidential-gke-nodes.

[195] Amazon Web Services, 'The security design of the aws nitro system: Aws whitepaper,' Tech. Rep., Feb. 2024, Accessed 2025-06-15. [Online]. Available: https://docs.aws.amazon.com/pdfs/whitepapers/latest/security-design-of-aws-nitro-system/security-design-of-aws-nitro-system.pdf.

[196] D. Brown. 'Confidential computing: An aws perspective.' Accessed 2025-06-15. (Aug. 2021), [Online]. Available: https://aws.amazon.com/blogs/security/confidential-computing-an-aws-perspective/.

[197] S. Shinde, D. T. T. Le, S. Tople and P. Saxena, 'Panoply: Low-TCB Linux Applications with SGX Enclaves,' in *Network and Distributed System Security Symposium (NDSS)*, 2017. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2017/09/ndss2017_07-5_Shinde_paper.pdf.

[198] M. Russinovich, *Azure and intel commit to delivering next generation confidential computing*, https://azure.microsoft.com/en-us/blog/azure-and-intel-commit-to-delivering-next-generation-confidential-computing/, Accessed July 2025, 2020.

[199] Decentriq. 'Revolutionising biomedical data collaboration and ai-driven healthcare solutions: Search launches.' Accessed 2025-06-15. (Oct. 2024), [Online]. Available: https://www.decentriq.com/article/revolutionising-biomedical-data-sharing-and-ai-driven-healthcare-solutions-search-launches.

[200] M. Hagland. 'Ucsf health's ai breakthrough: What's been learned so far.' Accessed 2025-06-15. (Jan. 2022), [Online]. Available: https://www.hcinnovationgroup.com/analytics-ai/artifical-intelligence-machine-learning/article/21254285/ucsf-healths-ai-breakthrough-whats-been-learned-so-far.

[201] D. Buckley and C. Lio, *Case study repository: Privacy enhancing technologies in official statistics*, United Nations Statistics Division Wiki, Accessed 2025-06-13, 2024. [Online]. Available: https://unstats.un.org/wiki/spaces/UGTTOPPT/pages/150012018/Case+study+repository.

[202] R. Society and B. Academy, 'From privacy to partnership: Unlocking data for better results,' The Royal Society, 2023. [Online]. Available: https://royalsociety.org/-/media/policy/projects/privacy-enhancing-technologies/from-privacy-to-partnership.pdf.

[203] Intel Corporation, *Intel® Software Guard Extensions Developer Guide*, Revision 2.26, 2025. [Online]. Available: https://download.01.org/intel-sgx/latest/linux-latest/docs/Intel_SGX_Developer_Guide.pdf.

[204] Intel Corporation, *Intel® Software Guard Extensions Developer Reference for Linux OS*, Revision 2.26, Open Source, 2025. [Online]. Available: https://download.01.org/intel-sgx/latest/linux-latest/docs/Intel_SGX_Developer_Reference_Linux_2.26_Open_Source.pdf.

[205] Intel Corporation, *Intel® 64 and ia-32 architectures software developer's manual, volume 3d: System programming guide, part 4*, Order Number: 332831-087US. Refer to all ten volumes for full context., Mar. 2025. [Online]. Available: https://cdrdv2.intel.com/v1/dl/getContent/671269.

[206] C.-C. Tsai, D. E. Porter and M. Vij, 'Graphene-sgx: A practical library os for unmodified applications on sgx,' in *Proceedings of the 2017 USENIX Conference on Usenix Annual Technical Conference*, ser. USENIX ATC '17, Santa Clara, CA, USA: USENIX Association, 2017, pp. 645–658, ISBN: 9781931971386.

[207] J. Lind, C. Priebe, D. Muthukumaran, D. O'Keeffe, P.-L. Aublin, F. Kelbert, T. Reiher, D. Goltzsche, D. Eyers, R. Kapitza, C. Fetzer and P. Pietzuch, 'Glamdring: Automatic application partitioning for intel SGX,' in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, Santa Clara, CA: USENIX Association, Jul. 2017, pp. 285–298, ISBN: 978-1-931971-38-6. [Online]. Available: https://www.usenix.org/conference/atc17/technical-sessions/presentation/lind.

[208] H. Wang, P. Wang, Y. Ding, M. Sun, Y. Jing, R. Duan, L. Li, Y. Zhang, T. Wei and Z. Lin, 'Towards memory safe enclave programming with rust-sgx,' in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19, London, United Kingdom: Association for Computing Machinery, 2019, pp. 2333–2350, ISBN: 9781450367479. DOI: 10.1145/3319535.3354241. [Online]. Available: https://doi.org/10.1145/3319535.3354241.

[209] S. Mofrad, F. Zhang, S. Lu and W. Shi, 'A comparison study of intel sgx and amd memory encryption technology,' in *Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy*, ser. HASP '18, Los Angeles, California: Association for Computing Machinery, 2018, ISBN: 9781450365000. DOI: 10.1145/3214292.3214301. [Online]. Available: https://doi.org/10.1145/3214292.3214301.

[210] Y. Shen, H. Tian, Y. Chen, K. Chen, R. Wang, Y. Xu, Y. Xia and S. Yan, 'Occlum: Secure and efficient multitasking inside a single enclave of intel sgx,' in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '20, Lausanne, Switzerland: Association for Computing Machinery, 2020, pp. 955–970, ISBN: 9781450371025. DOI: 10.1145/3373376.3378469. [Online]. Available: https://doi.org/10.1145/3373376.3378469.

[211] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'Keeffe, M. L. Stillwell, D. Goltzsche, D. Eyers, R. Kapitza, P. Pietzuch and C. Fetzer, 'Scone: Secure linux containers with intel sgx,' in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'16, Savannah, GA, USA: USENIX Association, 2016, pp. 689–703, ISBN: 9781931971331.

[212] J. Rutkowska, *Introducing graphene-ng: Running arbitrary payloads in sgx enclaves*, Blog post, Jun. 2018. [Online]. Available: https://blog.invisiblethings.org/2018/06/11/graphene-ng.html.

[213] C. Zhao, D. Saifuding, H. Tian, Y. Zhang and C. Xing, 'On the performance of intel sgx,' in *2016 13th Web Information Systems and Applications Conference (WISA)*, 2016, pp. 184–187. DOI: 10.1109/WISA.2016.45.

[214] N. Weichbrodt, P.-L. Aublin and R. Kapitza, 'Sgx-perf: A performance analysis tool for intel sgx enclaves,' in *Proceedings of the 19th International Middleware Conference*, ser. Middleware '18, Rennes, France: Association for Computing Machinery, 2018, pp. 201–213, ISBN: 9781450357029. DOI: 10.1145/3274808.3274824. [Online]. Available: https://doi.org/10.1145/3274808.3274824.

[215] M. El-Hindi, T. Ziegler, M. Heinrich, A. Lutsch, Z. Zhao and C. Binnig, 'Benchmarking the second generation of intel sgx hardware,' in *Proceedings of the 18th International Workshop on Data Management on New Hardware*, ser. DaMoN '22, Philadelphia, PA, USA: Association for Computing Machinery, 2022, ISBN: 9781450393782. DOI: 10.1145/3533737.3535098. [Online]. Available: https://doi.org/10.1145/3533737.3535098.

[216] J. Z. Yu, S. Shinde, T. E. Carlson and P. Saxena, 'Elasticlave: An efficient memory model for enclaves,' in *31st USENIX Security Symposium (USENIX Security 22)*, Boston, MA: USENIX Association, Aug. 2022, pp. 4111–4128, ISBN: 978-1-939133-31-1. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/yu-jason.

[217] G. E. P. Box and N. R. Draper, *Empirical Model-Building and Response Surfaces*. Wiley, 1987, p. 424, ISBN: 978-0471810339.

[218] J. Rutkowska and R. Wojtczuk, *Qubes OS architecture*, https://www.qubes-os.org/attachment/doc/arch-spec-0.3.pdf, Jan. 2010.

[219] Intel Corporation, *Intel® Software Guard Extensions (Intel® SGX) DCAP ECDSA Orientation Guide*, https://download.01.org/intel-sgx/latest/dcap-latest/linux/docs/DCAP_ECDSA_Orientation.pdf, 2020.

[220] V. Scarlata, S. Johnson, J. Beaney and P. Zmijewski, 'Supporting third party attestation for intel® sgx with intel® data center attestation primitives,' 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:221506554.

[221] M. U. Sardar, R. Faqeh and C. Fetzer, 'Formal foundations for intel sgx data center attestation primitives,' in *Formal Methods and Software Engineering*, S.-W. Lin, Z. Hou and B. Mahony, Eds., Cham: Springer International Publishing, 2020, pp. 268–283, ISBN: 978-3-030-63406-3.

[222] I. Corporation, *Intel® tdx dcap quoting library api documentation*, https://download.01.org/intel-sgx/latest/dcap-latest/linux/docs/Intel_TDX_DCAP_Quoting_Library_API.pdf, 2025.

[223] T. Knauth, M. Steiner, S. Chakrabarti, L. Lei, C. Xing and M. Vij, *Integrating remote attestation with transport layer security*, 2019. arXiv: 1801.05863 [cs.CR]. [Online]. Available: https://arxiv.org/abs/1801.05863.

[224] H. Tschofenig, Y. Sheffer, P. Howard, I. Mihalcea, Y. Deshpande, A. Niemi and T. Fossati, 'Using Attestation in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS),' Internet Engineering Task Force, Internet-Draft draft-fossati-tls-attestation-09, Apr. 2025, Work in Progress, 34 pp. [Online]. Available: https://datatracker.ietf.org/doc/draft-fossati-tls-attestation/09/.

[225] M. Kowalczyk, D. Kuvaiskii, P. Marczewski, B. Popławski, D. Porter, K. Qin, C.-C. Tsai, M. Vij and I. Yamahata, *Rapid deployment of confidential cloud applications with gramine*, ACSAC 2024 Artifacts Competition Finalist, 2024. [Online]. Available: https://www.acsac.org/2024/program/artifacts_competition/comp-acsac24-final10.pdf.

[226] Intel Corporation, 'Enclave-to-enclave communication in intel® software guard extensions (intel® sgx) applications,' Intel Corporation, White Paper 671547, 2019, Published April 22, 2019. [Online]. Available: https://software.intel.com/en-us/sgx.

[227] D. Vatsalan, Z. Sehili, P. Christen and E. Rahm, 'Privacy-preserving record linkage for big data: Current approaches and research challenges,' in *Handbook of Big Data Technologies*, A. Y. Zomaya and S. Sakr, Eds. Cham: Springer International Publishing, 2017, pp. 851–895, ISBN: 978-3-319-

49340-4. DOI: 10.1007/978-3-319-49340-4_25. [Online]. Available: https://doi.org/10.1007/978-3-319-49340-4_25.

[228] Intel Corporation, *Which intel trusted execution environment is right for you?* Infographic, ID: 818845. Learn which things to consider when choosing whether Intel TDX or Intel SGX is the right Intel Trusted Execution Environment for you, Mar. 2024. [Online]. Available: https://www.intel.com/content/www/us/en/content-details/818845/which-intel-trusted-execution-environment-is-right-for-you.html (visited on 15/05/2025).

[229] D. Lee, D. Jung, I. T. Fang, C.-C. Tsai and R. A. Popa, 'An off-chip attack on hardware enclaves via the memory bus,' in *Proceedings of the 29th USENIX Conference on Security Symposium*, ser. SEC'20, USA: USENIX Association, 2020, ISBN: 978-1-939133-17-5.

[230] C. Perezvargas. 'OpenHCL: The New, Open Source Paravisor.' Windows OS Platform Blog, Microsoft Tech Community. (Oct. 2024), [Online]. Available: https://techcommunity.microsoft.com/blog/windowsosplatform/openhcl-the-new-open-source-paravisor/4273172.

[231] N. Ferguson, B. Schneier and T. Kohno, 'The context of cryptography,' in *Cryptography Engineering*. John Wiley & Sons, Ltd, 2015, ch. 1, pp. 1–22, ISBN: 9781118722367. DOI: https://doi.org/10.1002/9781118722367.ch1. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781118722367.ch1. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781118722367.ch1.

[232] R. Cox, *The mos 6502 and the best layout guy in the world*, https://research.swtch.com/6502, Posted on Monday, January 3, 2011, Jan. 2011. [Online]. Available: https://research.swtch.com/6502.

[233] V. Costan, I. Lebedev and S. Devadas, *Sanctum: Minimal hardware extensions for strong software isolation*, Cryptology ePrint Archive, Paper 2015/564, 2015. [Online]. Available: https://eprint.iacr.org/2015/564.

[234] Department of Defense, 'Department of Defense Trusted Computer System Evaluation Criteria,' U.S. Department of Defense, Tech. Rep. DoD 5200.28-STD, 1985. [Online]. Available: https://csrc.nist.gov/files/pubs/conference/1998/10/08/proceedings-of-the-21st-nissc-1998/final/docs/early-cs-papers/dod85.pdf.

[235] D. Vanoverloop, A. Sanchez, F. Toffalini, F. Piessens, M. Payer and J. Van Bulck, 'TLBlur: Compiler-assisted automated hardening against controlled channels on off-the-shelf Intel SGX platforms,' in *34th USENIX Security Symposium (USENIX Security 25)*, Aug. 2025.

[236] M. Bölcskei, P. Jattke, J. Wikner and K. Razavi, 'Rubicon: Precise Microarchitectural Attacks with Page-Granular Massaging,' in *EuroS&P*, Jun. 2025. [Online]. Available: Paper=https://comsec.ethz.ch/wp-content/files/rubicon_eurosp25.pdf.

[237] N. A. Simakov, M. D. Innus, M. D. Jones, J. P. White, S. M. Gallo, R. L. DeLeon and T. R. Furlani, 'Effect of meltdown and spectre patches on the performance of HPC applications,' *CoRR*, vol. abs/1801.04329, 2018. arXiv: 1801.04329. [Online]. Available: http://arxiv.org/abs/1801.04329.

[238] M. Larabel, *The brutal performance impact from mitigating the lvi vulnerability*, https://www.phoronix.com/review/lvi-attack-perf, Accessed: 2025-07-01, 2020.

[239] G. Dessouky, D. Gens, P. Haney, G. Persyn, A. Kanuparthi, H. Khattri, J. M. Fung, A.-R. Sadeghi and J. Rajendran, 'HardFails: Insights into Software-Exploitable hardware bugs,' in *28th USENIX Security Symposium (USENIX Security 19)*, Santa Clara, CA: USENIX Association, Aug. 2019, pp. 213–230, ISBN: 978-1-939133-06-9. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/dessouky.

[240] Intel Corporation, *Intel® xeon® d processor advisory: Intel-sa-01045*, https://www.intel.com/content/www/us/en/security-center/advisory/INTEL-SA-01045.html, Firmware vulnerability (CVE-2023-43490) affecting SGX-enabled Intel® Xeon® D Processors, Mar. 2024.

[241] J. Eads, T. Ormandy, M. Rizzo, K. Janke and E. V. Nava, *Zen and the art of microcode hacking*, https://bughunters.google.com/blog/5424842357473280/zen-and-the-art-of-microcode-hacking, Google Bug Hunters Blog, Mar. 2025.

[242] P. Borrello, C. Easdon, M. Schwarzl, R. Czerny and M. Schwarz, 'Customprocessingunit: Reverse engineering and customization of intel microcode,' in *2023 IEEE Security and Privacy Workshops (SPW)*, 2023, pp. 285–297. DOI: 10.1109/SPW59333.2023.00031.

[243] J. Butterworth, C. Kallenberg, X. Kovah and A. Herzog, 'Bios chronomancy: Fixing the core root of trust for measurement,' in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ser. CCS '13, Berlin, Germany: Association for Computing Machinery, 2013, pp. 25–36, ISBN: 9781450324779. DOI: 10.1145/2508859.2516714. [Online]. Available: https://doi.org/10.1145/2508859.2516714.

[244] M. Ermolov, *Last barrier destroyed, or compromise of fuse encryption key for intel security fuses*, https://swarm.ptsecurity.com/last-barrier-destroyed-or-compromise-of-fuse-encryption-key-for-intel-security-fuses/, Accessed: 2025-07-01, 2025.

[245] I. Corporation, *Intel sgx key disclosure claim: Official statement*, https://www.intel.com/content/www/us/en/security-center/announcement/intel-security-announcement-2024-08-29-001.html, Accessed: 2025-07-01, 2025.

[246] K. Yang, M. Hicks, Q. Dong, T. Austin and D. Sylvester, 'A2: Analog malicious hardware,' in *2016 IEEE Symposium on Security and Privacy (SP)*, 2016, pp. 18–37. DOI: 10.1109/SP.2016.10.

[247] Intel Corporation, *Intel® Xeon® Scalable Processors: NEX Eagle Stream Platform – Intel Platform Security*, Document Number: 784473, Aug. 2023. [Online]. Available: http://cdrdv2-public.intel.com/784473/784473_Intel%20Platform%20Security.pdf.

[248] Intel Corporation, *Supporting Intel® SGX on Multisocket Platforms*, White paper, Document ID: 843058. PDF retrieved 2025-06-12, Dec. 2024. [Online]. Available: https://www.intel.com/content/www/us/en/content-details/843058/supporting-intel-sgx-on-multisocket-platforms.html.

[249] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh and V. Shmatikov, 'The most dangerous code in the world: Validating ssl certificates in non-browser software,' in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12, Raleigh, North Carolina, USA: Association for Computing Machinery, 2012, pp. 38–49, ISBN: 9781450316514. DOI: 10.1145/2382196.2382204. [Online]. Available: https://doi.org/10.1145/2382196.2382204.

[250] J. Van Bulck, D. Oswald, E. Marin, A. Aldoseri, F. D. Garcia and F. Piessens, 'A tale of two worlds: Assessing the vulnerability of enclave shielding runtimes,' in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19, London, United Kingdom: Association for Computing Machinery, 2019, pp. 1741–1758, ISBN: 9781450367479. DOI: 10.1145/3319535.3363206. [Online]. Available: https://doi.org/10.1145/3319535.3363206.

[251] F. Alder, L.-A. Daniel, D. Oswald, F. Piessens and J. Van Bulck, 'Pandora: Principled symbolic validation of Intel SGX enclave runtimes,' in *45th IEEE Symposium on Security and Privacy (S&P)*, May 2024.

[252] J. Van Bulck, F. Alder and F. Piessens, 'A case for unified ABI shielding in Intel SGX runtimes,' in *5th Workshop on System Software for Trusted Execution (SysTEX)*, ACM, Mar. 2022.

[253] R. Krahn, D. Dragoti, F. Gregor, D. L. Quoc, V. Schiavoni, P. Felber, C. Souza, A. Brito and C. Fetzer, 'Teemon: A continuous performance monitoring framework for tees,' in *Proceedings of the 21st International Middleware Conference*, ser. Middleware '20, Delft, Netherlands: Association for Computing Machinery, 2020, pp. 178–192, ISBN: 9781450381536. DOI: 10.1

145/3423211.3425677. [Online]. Available: https://doi.org/10.1145/3423211.3425677.

[254] I. Corporation, *Q1'2024 intel sgx trusted computing base (tcb) recovery guidance*, https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/resources/q1-2024-intel-tcb-recovery-guidance.html, Provides details on TCB Recovery enforcement, required updates, and attestation collateral for Intel SGX platforms., Mar. 2024.

[255] L. Abdullah, F. Freiling, J. Quintero and Z. Benenson, 'Sealed computation: Abstract requirements for mechanisms to support trustworthy cloud computing,' in *Computer Security*, S. K. Katsikas, F. Cuppens, N. Cuppens, C. Lambrinoudakis, A. Antón, S. Gritzalis, J. Mylopoulos and C. Kalloniatis, Eds., Cham: Springer International Publishing, 2019, pp. 137–152, ISBN: 978-3-030-12786-2.

[256] A. Trask, E. Bluemke, T. Collins, B. G. E. Drexler, C. G. Cuervas-Mons, I. Gabriel, A. Dafoe and W. Isaac, *Beyond privacy trade-offs with structured transparency*, 2024. arXiv: 2012.08347 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2012.08347.

[257] M. Schwarz and D. Gruss, 'How trusted execution environments fuel research on microarchitectural attacks,' *IEEE Security & Privacy*, vol. 18, no. 5, pp. 18–27, 2020. DOI: 10.1109/MSEC.2020.2993896.

[258] C. Cohen, J. Forshaw, J. Horn and M. Brand, 'Amd secure processor for confidential computing security review,' Technical Report. Google Project Zero and Google Cloud Security, Tech. Rep., 2022. [Online]. Available: https://storage.googleapis.com/gweb-uniblog-publish-prod/documents/AMD_GPZ-Technical_Report_FINAL_05_2022.pdf.

[259] R. Branco, *The microarchitectures that i saw and the ones that i hope to one day see*, Keynote presentation at Hardwear.io USA 2023, Slides: https://hardwear.io/usa-2023/presentation/the-microarchitectures-that-I-saw-and-the-ones-that-I-hope-to-one-day-see.pdf, Video: https://www.youtube.com/watch?v=WlcQrx7VK00, Jun. 2023.

[260] R. M. Koduri, *Intel keynote at hotchips 32*, https://hc32.hotchips.org/assets/program/conference/day1/HC32-K1-Intel-Raja-2020.pdf, HotChips 32 Conference, 2020.

[261] S. Volos, C. Fournet, J. Hofmann, B. Köpf and O. Oleksenko, 'Principled microarchitectural isolation on cloud cpus,' in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '24, Salt Lake City, UT, USA: Association for Computing Machinery, 2024, pp. 183–197, ISBN: 9798400706363. DOI: 10.1145/3658644.3690183. [Online]. Available: https://doi.org/10.1145/3658644.3690183.

[262] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanović and D. Song, 'Keystone: An open framework for architecting trusted execution environments,' in *Proceedings of the Fifteenth European Conference on Computer Systems*, ser. EuroSys '20, Heraklion, Greece: Association for Computing Machinery, 2020, ISBN: 9781450368827. DOI: 10.1145/3342195.3387532. [Online]. Available: https://doi.org/10.1145/3342195.3387532.

[263] M. Kuhne, S. Volos and S. Shinde, *Dorami: Privilege separating security monitor on risc-v tees*, 2024. arXiv: 2410.03653 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2410.03653.

[264] E. Feng, X. Lu, D. Du, B. Yang, X. Jiang, Y. Xia, B. Zang and H. Chen, 'Scalable memory protection in the PENGLAI enclave,' in *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, USENIX Association, Jul. 2021, pp. 275–294, ISBN: 978-1-939133-22-9. [Online]. Available: https://www.usenix.org/conference/osdi21/presentation/feng.

[265] J. Noorman, J. Van Bulck, J. T. Mühlberg, F. Piessens, P. Maene, B. Preneel, I. Verbauwhede, J. Götzfried, T. Müller and F. Freiling, 'Sancus 2.0: A low-cost security architecture for IoT devices,' *ACM Transactions on Privacy and Security (TOPS)*, vol. 20, no. 3, 7:1–7:33, Sep. 2017.

[266] G. Scopelliti, S. Pouyanrad, J. Noorman, F. Alder, F. Piessens and J. T. Mühlberg, 'Poster: An open-source framework for developing heterogeneous distributed enclave applications,' in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 2393–2395.

[267] J. T. Mühlberg and J. Van Bulck, 'Reflections on post-Meltdown trusted computing: A case for open security processors,' *;login: the USENIX magazine*, vol. Vol. 43, no. No. 3, Fall 2018.

[268] C. Stoller and M. Osorio, 'Fabrication begins for production opentitan silicon,' *Google Open Source Blog*, Feb. 2025, The latest news from Google on open source releases, major projects, events, and student outreach programs.

[269] Accenture and Amazon Web Services, 'How moving onto the aws cloud reduces carbon emissions,' Accenture and Amazon Web Services, Tech. Rep., 2024, [Online]. Available: https://sustainability.aboutamazon.com/carbon-reduction-aws.pdf.

[270] E. Masanet, A. Shehabi, L. Ramakrishnan, J. Liang, X. Ma, B. Walker, V. Hendrix and P. Mantha, 'The energy efficiency potential of cloud-based software: A u.s. case study,' Lawrence Berkeley National Laboratory, Tech. Rep., 2013.

[271] C. P. Baldé, V. Forti, V. Gray, R. Kuehr and P. Stegmann, *The global e-waste monitor 2017: Quantities, flows and resources*. United Nations University, international telecommunication union, and ..., 2017.

[272] International Telecommunication Union, *Goal 12: Sustainable consumption & production (icts for a sustainable world)*, https://www.itu.int/en/sustainable-world/pages/goal12.aspx, 2023.

[273] United Nations Statistics Division, *Sdg 16: Access to information and protect fundamental freedoms (target 16.10)*, https://unstats.un.org/sdgs/metadata/?Goal=16, 2023.

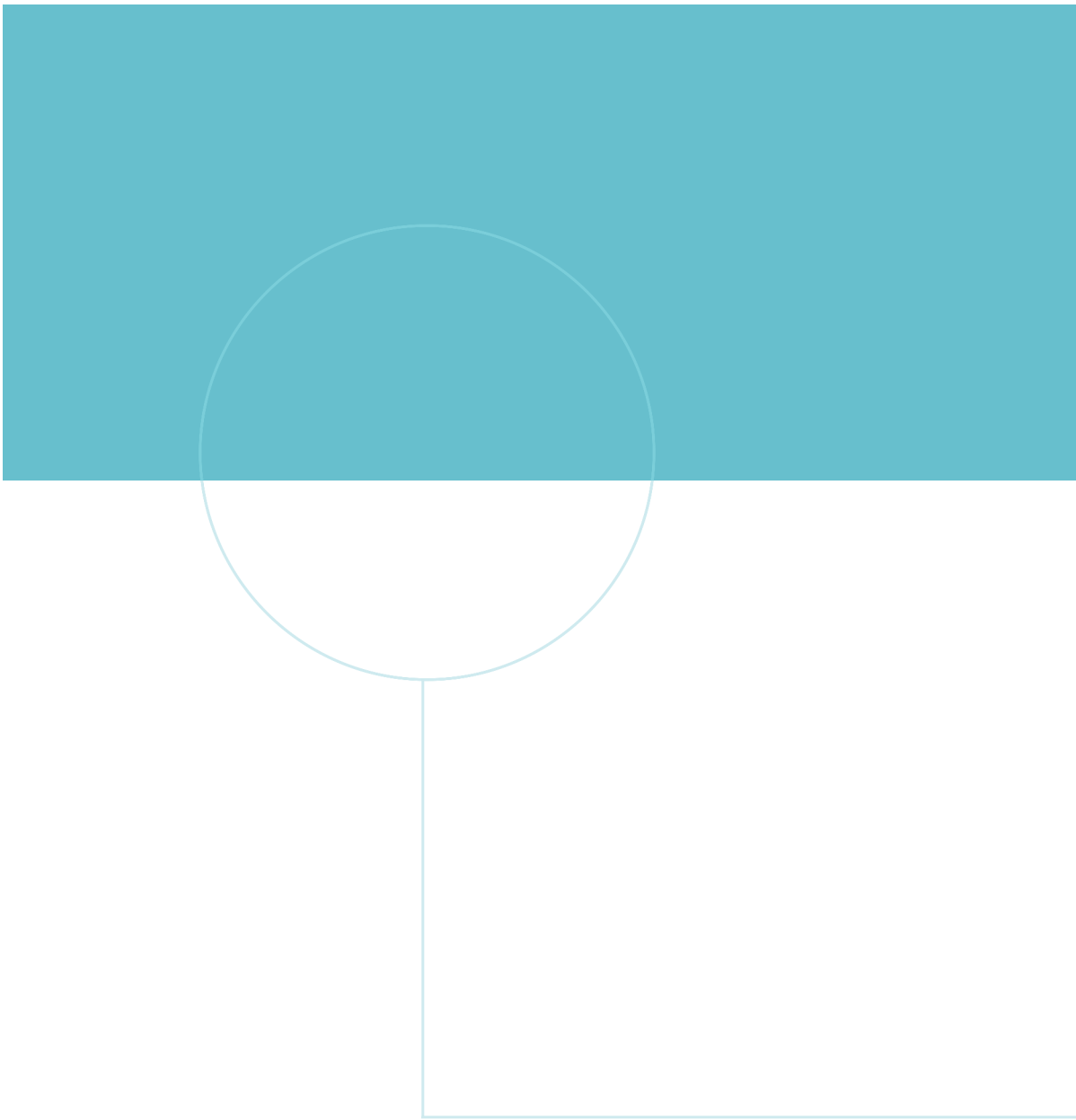
[274] UNESCO, *From Promise to Practice: Access to Information for Sustainable Development, 2020 UNESCO Report on the Monitoring and Reporting of SDG Indicator 16.10.2 (Public Access to Information)*. Paris: United Nations Educational, Scientific and Cultural Organization, 2020, [Online]. Available: https://unesdoc.unesco.org/ark:/48223/pf0000375022, ISBN: 978-92-3-100421-6.

[275] U. OHCHR, 'A human rights-based approach to data. leaving no one behind in the 2030 agenda for sustainable development,' *United Nations, Geneva, Switzerland*, 2018.

[276] F. N. Wirth, T. Meurers, M. Johns and F. Prasser, 'Privacy-preserving data sharing infrastructures for medical research: Systematization and comparison,' *BMC Medical Informatics and Decision Making*, vol. 21, pp. 1–13, 2021.

[277] G. S. D. R. 2023, *Times of crisis, times of change: Science for accelerating transformations to sustainable development*, 2023. [Online]. Available: https://sdgs.un.org/sites/default/files/2023-09/FINAL%20GSDR%202023-Digital%20-110923_1.pdf.

[278] I. Security, 'Confidential computing and the fight to end modern slavery,' Intel, Tech. Rep., 2022, Solution Brief. [Online]. Available: https://www.intel.com/content/dam/www/central-libraries/us/en/documents/2022-12/hope-for-justice-solution-brief.pdf.



NTNU

Norwegian University of
Science and Technology