Secure and Policy-Compliant Query Processing on Heterogeneous Computational Storage Architectures

Harshavardhan Unnibhavi
Technical University of Munich

David Cerdeira
Centro ALGORITMI,
University of Minho

Antonio Barbalace
The University of Edinburgh

Nuno Santos
INESC-ID / Instituto Superior Técnico,
Universidade de Lisboa

Pramod Bhatotia
Technical University of Munich

Abstract

Computation Storage Architectures (CSA) are increasingly adopted in the cloud for near data processing, where the underlying storage devices/servers are now equipped with heterogeneous cores which enable computation offloading near to the data. While CSA is a promising high-performance architecture for the cloud, in general data analytics also presents significant data security and policy compliance (e.g., GDPR) challenges in untrusted cloud environments.

In this paper, we present IronSafe, a secure and policy-compliant query processing system for heterogeneous computational storage architectures, while preserving the performance advantages of CSA in untrusted cloud environments. To achieve these design properties in a computing environment with heterogeneous host (x86) and storage system (ARM), we design and implement the entire hardware and software system stack from the ground-up leveraging hardware-assisted Trusted Execution Environments (TEEs): namely, Intel SGX and ARM TrustZone. More specifically, IronSafe builds on three core contributions: (1) a heterogeneous confidential computing framework for shielded execution with x86 and ARM TEEs and associated secure storage system for the untrusted storage medium; (2) a policy compliance monitor to provide a unified service for attestation and policy compliance; and (3) a declarative policy language and associated interpreter for concisely specifying and efficiently evaluating a rich set of polices. Our evaluation using the TPC-H SQL benchmark queries and GDPR anti-pattern use-cases shows that IronSafe is faster, on average by 2.3× than a host-only secure system, while providing strong security and policy-compliance properties.

CCS Concepts

• **Security and privacy → Trusted computing**; **Database and storage security**; **Information accountability and usage control**.

ACM Reference Format:
Harshavardhan Unnibhavi, David Cerdeira, Antonio Barbalace, Nuno Santos, and Pramod Bhatotia. 2022. Secure and Policy-Compliant Query Processing on Heterogeneous Computational Storage Architectures. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22), June 12–17, 2022, Philadelphia, PA, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3514221.3517913

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SIGMOD '22, June 12–17, 2022, Philadelphia, PA, USA
© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9249-5/22/06…$15.00
https://doi.org/10.1145/3514221.3517913

1 Introduction

Today, companies are increasingly turning to cloud providers to run their data management services with improved scalability, reliability, and cost-effectiveness [21, 24, 56]. Consequently, the data stored in cloud environments is growing at an ever-increasing rate [1, 2, 33]. To handle this data deluge, the cloud systems are advancing the state of hardware [5] and software [3, 9] to enable high-performance query processing on large volumes of data.

While *scalability* and *performance* have always been at the center-stage in designing query processing engines, we are now increasingly facing pressing challenges to ensure the *security* and *policy compliance* of stored data and query processing in cloud environments [12, 68, 96, 115]. In particular, security has become a central priority as a result of concerns about the untrusted nature of the cloud [104]. Despite the growing sophistication of defenses, the complexity of existing cloud computing and storage systems is staggering, which means that the existence of vulnerabilities is inevitable. Attackers may attempt to breach into the systems in various ways, either through an external vector, e.g., by exploiting software bugs or, configuration errors [7], or via an internal vector, e.g., through, malicious admins [8, 85], or employees victim of social engineering attacks [111]. These threats are even more pronounced due to virtualization in the cloud, where a co-located tenant [107] or even a malicious cloud administrator [106] can violate the confidentiality and integrity properties.

Likewise, policy compliance is the second major issue. Even considering the existence of a foolproof security infrastructure that can protect the data and computations from intruders, the way data can be collected, stored, processed, and shared by organizations is today highly regulated by national and international laws. Some notable examples of data protection regulations include GDPR in the EU [101] and CCPA in the USA [38]. Failure to abide to these and similar laws when processing personal data may lead to the payment of high fees [53]. As a result, query processing engines also need to incorporate built-in mechanisms that can help reduce the risks of non compliance and give organizations the ability to tightly control how they store, process, and share data in third-party cloud environments.

In light of the aforementioned considerations, in this paper we focus on this question: *How can we build a high-performance query processing system that enables organizations to process and share high-value data with strong security properties against powerful adversaries and policy compliance guarantees in third-party cloud environments?*

To answer this question, our key idea is to leverage two prominent advances in hardware technology, both of them now readily available on commodity cloud infrastructures: (a) *Computational Storage Architecture* (CSA), and (b) *Trusted Execution Environments* (TEEs).

First, CSA helps us to reap the performance benefits of offloading query processing jobs near the actual data location, aka, near data processing [18, 87]. CSA platforms are increasingly heterogeneous: with x86 hosts and storage systems mounting low-power ARM [28, 59, 63, 87], which can be exploited to move parts of query processing (e.g., filter/select operations [28, 59]) or storage operations (e.g., checksum/deduplication [10, 33]) near to the data. By offloading the computation directly onto the storage system, CSA strives for increased performance since it minimizes the data movement across multiple hardware and software layers [28, 29, 46, 63]. CSA is increasingly adopted in cloud environments, both for server-class and SSD devices storage systems. For example, Huawei Cloud [63] and Microsoft Azure [78] are designing CSA storage-class servers based on ARM. Two major startups are also offering ARM-based CSA servers: NGD systems [87] and SoftIron [39]. For storage devices Samsung [112] offers ARM-based/FPGA-based SSDs.

Second, TEEs help us build a security infrastructure that can protect data and computations against powerful adversaries. TEEs are rooted in trusted hardware components and expose several basic primitives that can be used for designing secure systems. One of the fundamental primitives enables the creation of isolated memory regions and execution contexts (commonly known as *enclaves*) where code and data are protected by the CPU, even against the privileged code (e.g., OS, hypervisor). Based on this promise, TEE technology is now being streamlined by all major CPU manufacturers, e.g., Intel SGX [66], ARM TrustZone [20], Realms in ARM v9 [6], AMD SEV [4], and Keystone in RISC-V [102]. Likewise, all major cloud providers have started harnessing TEE technology to offer confidential computing services (AliBaba, Azure, IBM, and Google).

Our approach is then to leverage both these techniques to build a full-blown secure and policy compliant query processing system. Achieving this goal, however, is not straightforward. Succinctly stated, we have to overcome three main technical challenges. First, CSA architectures are equipped with heterogeneous TEE technologies, such as host and storage device/server are equipped with Intel SGX [66] and ARM TrustZone [20], respectively, which offer fundamentally different and thus incompatible protection mechanisms. Therefore, we need to build an end-to-end security infrastructure that can bridge these incompatibilities and be able to both (i) shield the data computation involved in the various CSA query processing stages, and (ii) secure the data at rest on persistent storage. Second, to generate proofs of policy compliance, we can harness the remote attestation primitives provided by TEE technology, but they alone are insufficient to offer these proofs as they can only provide guarantees of integrity and authenticity of a runtime environment. They need to be complemented with additional semantically-richer mechanisms customized for query processing, and capable of coping with the dynamic and heterogeneous nature of cloud infrastructures. Third, we need to provide a declarative language that combines (a) simplicity in the specification (it must be simple to write and to interpret by different parties, (b) expressive enough to cover relevant regulatory-compliant use cases, and (c) efficient to evaluate.

Contributions. We present IronSafe, a secure and policy compliant query processing system. It is the first system that combines two emerging cloud technologies—CSA and TEE—for accelerating data processing while ensuring end-to-end security of query processing and data storage, respectively. IronSafe allows its users to specify data processing policies via a declarative language and obtain proofs of compliance. This service can be used, e.g., to enable different organizations to share data while offering hard-evidence of regulatory compliance. By addressing the challenges stated above in our IronSafe design, we make the following additional contributions:
(1) *Heterogeneous confidential computing framework:* To enable secure execution across the host and storage system, we develop a shielded execution framework across the CSA components that overcomes the technical hurdles due to the heterogeneity of TEE technologies. To protect data at rest, IronSafe includes a secure storage framework that builds on ARM TrustZone hardware features and ensures confidentiality, integrity, and freshness of stored data.
(2) *Policy compliance monitor:* We present the design of a policy compliance infrastructure based on a unified abstraction for attestation and policy enforcement, wherein a remote user can efficiently verify the authenticity of the host and storage systems to which the computation is offloaded and ensure that the computations are executed according to user-defined policies. Our design makes use of a supervising entity named *trusted monitor* that verifies the integrity and authenticity of IronSafe's query processing infrastructure, and coordinates the enforcement of policies across the system.
(3) *Declarative policy specification language:* IronSafe provides a unified declarative policy language to efficiently express a wide range of execution policies (e.g., GDPR) regarding data integrity, confidentiality, access accounting, and many other workflows. The declarative abstraction is imperative to minimize the complexity in specifying and auditing policies via a single unified interface.

We have implemented these components in an end-to-end system, from the ground-up (hardware and software). To show the effectiveness of the IronSafe system architecture, we have built a CSA database engine, which exposes a declarative (SQL) query and associated execution policy interface for GDPR compliance. We have evaluated the IronSafe database engine using the TPC-H benchmark [120], and a series of microbenchmarks to show the cost of each individual system components. Further, we have implemented and evaluated GDPR-antipatterns [116] using our declarative policy language. Our evaluation shows that IronSafe is faster, on average by 2.3× than a host only secure system, while achieving strong security and policy-compliance properties.

2 Background

2.1 Data Security and Policy Compliance

Data security concerns are evermore present in online services leveraging cloud computing and storage. Due to the sensitive nature of the data these services process and store (e.g., customer data), it is imperative to preserve the confidentiality and integrity of data and computation in untrusted cloud environments [35].

At the same time, recent legislative efforts for policy compliance (e.g., CCPA [38] and GDPR [101]) establish new rights and obligations regarding the use of personal data. For instance, GPDR [101, 117] describes four entities: (i) *data owner*, the person whose personal data is collected (i.e., customer), (ii) *controller*, which performs data

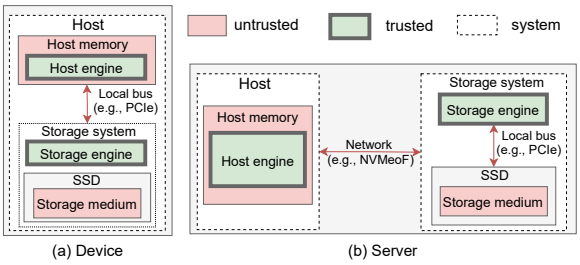


Figure 1: CSA deployment models: (a) device and (b) server

collection, (iii) *processor*, which processes personal data on behalf of the controller, (iv) *regulator*, which represents the supervisory authorities that oversee the compliance of rights and responsibilities to GDPR. *Data processing* must follow a set of principles, including storing data for a specific amount of time, or only using data for allowed purposes. *Rights of data owners* allow people, e.g., to know the purpose their data will serve, and for exactly how long it will be used. *Data controllers* set up secure infrastructure, maintain records of data processing, control the location of data, and establish interfaces so that people can exercise their rights.

2.2 Computational Storage Architectures

Computational Storage Architectures (CSA) provide a flavour of Near Data Processing (NDP) that splits a computation among two processing systems: *host* or *compute node*—likely x86, and a *storage device or node*, co-located with storage medium (see Figure 1). Storage systems are increasingly include low-power CPUs, prominently ARM, such as modern SSDs [60, 87] or storage servers [39, 63, 78]. Hence, a system integrating CSA is likely a heterogeneous compute platform. By splitting computation and offloading part of data processing to the storage system, CSA minimizes data movements through the storage interconnect, i.e., network or local bus (PCIe) [28, 47]. Therefore, CSA have been adopted by the database community for a while, more recently, pushing down part of a query to computational storage servers [15, 93], or to computational storage devices [59, 69].

For CSAs based on a storage device, the processing units are located on a storage device itself, like an SSD, physically attached to the storage media (flash memory). In turn the storage device is accessed by a host engine via a local bus, such as PCIe [87, 88, 112], Figure 1 (a). Another flavor of CSAs is based on a storage server where system components execute in two different nodes: the host—running the host engine, and the storage server—where the near data processing takes place, atop a storage engine. See Figure 1 (b). This resembles a cloud data-center deployment entailing compute and storage servers interconnected by high-speed network (e.g., NVMe-oF [89], NFS).

2.3 Confidential Computing and TEE

The need to protect data while in use, gives rise to confidential computing. Solutions for securing data while at rest and while moving, exist and are often used (e.g., TLS, and File Disk Encryption), however protecting data while in use requires the use of Trusted Execution Environments (TEEs). TEE technology offers an environment which is shielded from outside interference, and provides the necessary mechanisms to build security-sensitive applications. Prominent examples include Intel SGX [66], ARM TrustZone [20] and ARM v9 Realms [6]. ARM v9 Realms, introduced recently, aims to provide containerized execution environments that are protected against to the privileged software layers, including the OS or the hypervisor.

In this work, we build on ARM TrustZone, and Intel SGX. Importantly, we need to consider that these TEEs provide starkly different security properties, usage scenarios, implementation, and programming models. TrustZone provides two hardware-assisted security domains: the *normal world*, also referred to as Rich Execution Environment (REE), where a general purpose OS operates, and the *secure world* which typically hosts the TEE software comprising a trusted OS and multiple trusted applications (TA) [20]. Security features available to TAs include access to secure storage, monotonic counters, true random generators, trusted device, among others [118]. Secure boot allows for authentication of each software image, and allows the system to be brought to a known secure state. Intel SGX provides an abstraction of enclave—a memory region for which the CPU guarantees confidentiality and integrity. Enclave applications are protected against attacks launched by privileged software (e.g., OS or hypervisor). By virtue of an on-chip memory encryption engine, enclave memory is never stored in plain text to main memory. Additionally, SGX provides the features required to provided attestation of enclaves to third parties.

3 Overview

In this section, we describe the high-level architecture of IronSafe, our threat model, and the key design challenges.

3.1 Architecture and Workflow

The IronSafe architecture consists of the following components, as depicted in Figure 2: *client*, *trusted monitor*, *host* and *storage system*. Client is a piece of software that provides a data query interface that is used by different parties, including data producers and consumers. This interface offers the ability to require the enforcement of specific security policies when submitting queries to IronSafe's query processing infrastructure. The client interacts with the query processing infrastructure that in turn consists of two central components responsible for query processing—host and storage system—and a supervising component—the monitor. The host features an untrusted OS and a trusted host engine. The storage system includes the (trusted) storage engine, which is executed in a processing unit either within a storage device, or a storage server, and the (untrusted) storage medium where data is stored. The storage engine handles processing to be done near the data. The trusted monitor coordinates with the host engine and storage system to execute the computation or data operations in a policy compliant manner. The trusted monitor ensures this by attesting both the host and storage engine. The security of attestation is rooted on trusted boot and several other hardware-backed mechanisms [16].

To illustrate the workflow of IronSafe, consider the scenario depicted in Figure 2. It represents two entities $\mathcal{A}$ and $\mathcal{B}$ that collaborate with each other in processing data from a customer (C). Both these entities play the GDPR role of controllers as they offer to customers some specific service, e.g., $\mathcal{A}$ is an airline company and $\mathcal{B}$ is a hotel chain. In GDPR, customers retain ownership of their data. In this scenario, $\mathcal{A}$ collects data from its customers, e.g., when booking a flight, and it is willing to allow $\mathcal{B}$ to issue certain external queries, e.g., to consult the arrival time of a particular customer. IronSafe provides a secure storage infrastructure that allows $\mathcal{A}$ to share data with $\mathcal{B}$ in a secure and policy-compliant manner. In this sense, $\mathcal{A}$ and $\mathcal{B}$ act as data producer and consumer, respectively. IronSafe is deployed by a cloud provider who plays the role of GDPR data processor, and it allows both $\mathcal{A}$ and $\mathcal{B}$ to submit different queries: $\mathcal{A}$

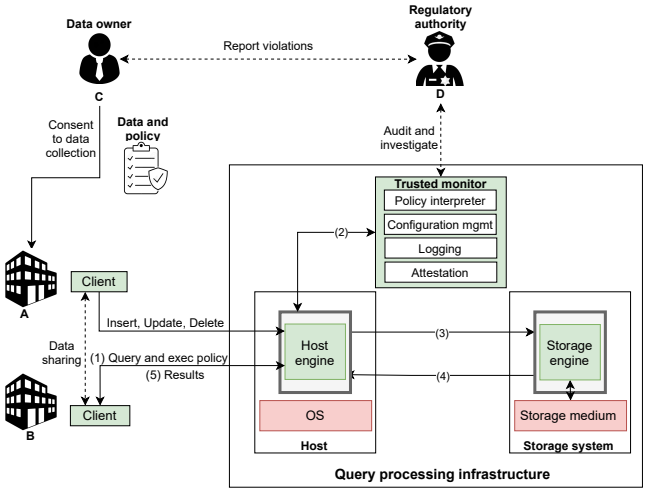


Figure 2: Architecture of IronSafe (TCB shown in green)

has full r/w access to the data being able to insert, update, or delete records; $\mathcal{B}$ is allowed only to perform certain read-only queries. The entire query processing workflow is determined by the compliance policies attached to each query. These policies can be specified so as to prevent violation of regulations such as the GDPR. Upon request, IronSafe allows a designated regulatory authority ($\mathcal{D}$) to obtain proof of compliance by accessing an audit trail from the trusted monitor.

More specifically, the workflow of any given data processing query is as follows. When instructed by $\mathcal{A}$ or $\mathcal{B}$, the client submits the query and an associated user-defined *execution policy* to the host engine over an encrypted TLS channel (step ①). Upon receiving a client request (i.e., query and corresponding policy), the host requests the assistance of the trusted monitor to verify that the client's request has the right permissions to access the data. This verification is performed by validating the query and the identity of the client against an *access policy* previously specified by the data producer. This access policy is maintained by the trusted monitor and it is set up by the time the data producer initializes the database on IronSafe. Before authorizing the execution of the query, the monitor must also check if the host and storage engine satisfy the constraints specified in the client's compliance policy. This is performed in step ②. If the check succeeds, the query is partitioned in the host engine, and offloaded to storage engine (step ③). The storage engine accesses users' data on persistent storage, executes the offloaded query and relays the results back to the host in step ④. The host engine process the filtered out data by executing its query. Then it sends to the client both the query results and proof of compliance with client's security policy (step ⑤). In IronSafe, both execution policy and access policy are specified in our own declarative policy specification language.

3.2 Hardware Building Blocks

Trusted execution environments. IronSafe requires TEE hardware technology for securing the execution of query processing both at the host and storage systems. We envision the host system to be an x86 machine. Thus, confidential execution will be supported by SGX enclaves [66]. As for the storage system, IronSafe benefits from bringing query execution to processing units resident as close as possible to the storage medium —where data is located. Given emerging computational storage devices are based on ARM CPUs, and storage servers increasingly adopt or include ARM CPUs (§ 2.2), the storage system's confidential execution relies primarily on ARM TrustZone [20].

Trusted boot. IronSafe relies on trusted boot for establishing its root of trust [19, 65]. Trusted boot validates the integrity and authenticity of the software that implements the host engine and storage system. In SGX, although the boot process is not part of the TCB, loading and initializing an enclave includes the process of computing a hash of the guest application code which allows remote parties to validate the correct initialization of an enclave. In TrustZone, trusted boot allows for authentication of each world's images, thus allowing for the system to be brought to a known secure state.

Remote attestation. IronSafe implements security protocols that allow a remote client to obtain hard cryptographic evidence about the correct deployment of all its components on a given cloud infrastructure. In SGX, remote attestation is supported by two hardware mechanisms which compute a hash of the enclave code upon initialization, and issue digital signatures of that hash using on-chip keys certified by Intel. TrustZone does not provide built-in remote attestation primitives apart from an on-chip key provided by the device manufacturer. In this case, we assume that remote attestation primitives are implemented by the device firmware running in the secure world.

Secure storage on the storage medium. IronSafe also relies on specific hardware primitives to implement secure storage on the storage medium. In TrustZone-based systems, it is common for flash memories to be used as storage medium. These flash memories, are often imbued with a partition which can only be access by an authenticated agent (i.e., a TEE) [50, 91]. This partition is a replay protected memory block (RPMB) and can then be used to hold security sensitive data. Additionally, TAs can store larger amounts of data by storing data in an encrypted form in the normal world file system [91].

3.3 Security Goals and Threat Model

We aim at designing IronSafe to provide end-to-end security for all data queries submitted and processed by our system, especially by host and storage system. Essentially, this protection entails preserving the confidentially, integrity, and freshness of all data at runtime (i.e., at processing time), at rest (i.e., on persistent storage), and in transit (i.e., exchanged over the network). In particular, we aim to defend against an adversary that can launch the following attacks:

Attacks to volatile state. As shown in Figure 2, the host engine and the storage engine need to process sensitive query-related data in main memory. We aim to defend against attacks aimed to inspect or tamper with the runtime state of these components. On the host, we consider an attacker with the ability to control the operating system. Targeting an x86 host equipped with SGX, the adversary will be able to access the full memory state, except the memory pages allocated to user-level enclaves; the attacker has no access to the enclave protecting the host engine. On the storage system, the attacker may also attempt to gain control of the storage engine runtime state (i.e., containing query processed data). Assuming that the storage system is deployed on a TrustZone-featured ARM platform and that the firmware (running in secure world) is trusted, the trusted boot mechanism will bring the storage system into a state that the attacker will not be able to control. This guarantee can also be extended to the REE OS (running in the normal world) as long as the REE OS is a secure component

(e.g., a hardened kernel) and trusted boot also measures the integrity of the REE OS. Nevertheless, the attacker may attempt to impersonate a trusted device so as to convince the host engine to offload computations to an alternative storage system controlled by the adversary.

Attacks to persistent state. In IronSafe, the persistent state is maintained on untrusted storage medium by the storage system. The attacker may attempt to bypass trusted boot by booting the storage system into an OS under its control and then gain unlimited access to the storage medium. At this point, the adversary may attempt to inspect the content of persistent data, or modify persistent data without being detected. In particular, it may attempt to launch rollback attacks in which the data on storage may be reverted to a stale version. Such an attack could result in the suppression of recently committed storage operations. We also consider forking attacks where the adversary can attempt to fork the storage system, by running multiple replicas of it.

Attacks to communications. An adversary may intercept, inject, or modify messages exchanged between all IronSafe's actors. The adversary's goals include: impersonation of legitimate actors, inspection of communications, or tampering with exchanged messages.

Attacks out of scope. We do not protect against side-channel attacks, such as exploiting cache timing, memory access patterns and speculative execution [36, 61, 128]. We also do not consider denial of service attacks as they are easy to defend with a trusted third party operator controlling the underlying infrastructure [30]. It is also out of scope physical attacks that involve tampering with the SGX or TrustZone hardware. These components are part of our trusted computing base. We also rule out sophisticated physical attacks to host and storage system, e.g., unsoldering SoC/CPU/flash chips. Attacks that exploit vulnerabilities in client applications and lead to data breaches through crafted queries, such as SQL injection, are correctly recorded into a tamper proof log by IronSafe. This logging process cannot be circumvented by an attacker as we trust the underlying hardware to protect the software responsible for logging(§ 4.3).

Due to the limitations of ARM TrustZone, we currently need to consider the entire OS stack and query engine on the storage side as part of our TCB. However, ARM v9 [6] aims to overcome this limitation, which would allow us to not trust the OS stack anymore.

3.4 Design Challenges

#1. Heterogeneous TEE technology. In IronSafe, to build our end-to-end security infrastructure, we rely on basic hardware primitives offered by SGX and TrustZone, namely TEE-isolated environments for securing in-memory state, and secure storage primitives for protection on-storage data (see § 3.2). However, the TEE primitives implemented by these technologies have important differences. For instance, SGX allows for seamlessly creating user-level enclaves, but they limit the amount of memory that can be allocated by applications and impose considerable performance overheads due to hardware-level memory encryption. On the other hand, TrustZone imposes no strict memory size limits and adds negligible performance overheads, but features a cumbersome protection model, where TEE environments are typically not available for general use by full-blown applications. Bringing together these two technologies while preserving security and performance is not a trivial task in the design of IronSafe. A second challenge involves the protection of data at rest. Although SGX and TrustZone provide protection for in-memory state, the mechanisms do not naturally extend to untrusted storage medium. To address both these challenges, we developed a *heterogeneous confidential computing framework* (see § 4.1).

#2: Policy compliance. In IronSafe, our approach to providing proof of policy compliance is twofold. First, we need to offer guarantees of full integrity and authenticity of all the components of IronSafe's TCB, which includes the host engine, the storage engine, and the trusted monitor. This guarantees the integrity of all the protocols responsible for policy enforcement and query processing. Second, we need to tie that proof of integrity and authenticity to each specific query. Only then we can offer guarantee that each specific query has been processed in a fully compliant manner. To achieve this goal, our starting point is to leverage the remote attestation primitives (see § 3.2) available on the core components of IronSafe's infrastructure. However, it is necessary to generate consistent attestation quotes across multiple components, i.e., those responsible for evaluating the security policy—the monitor—and those responsible for processing the query— host and storage engine. Achieving this goal is prone to security flaws, especially if we consider a cloud environment, where migration across servers occurs often, and multiple software versions and hardware generations co-exist. A second hurdle is due to an incompatibility between attestation mechanisms provided by TEE technologies. With SGX relying on IAS [16, 65], and TrustZone relying on customized remote attestation protocols implemented by secure world firmware and on manufacturer-dependent platform keys. Our proposed solution is a *policy compliance monitor* (§ 4.2).

#3: Policy specification. In IronSafe, we need to provide a declarative language for the specification of policies. These policies will allow data providers and data consumers to express their requirements for the execution of data processing queries (see § 3.1). At the same time, given the existence of important data protection regulations such as GDPR and CCPA, our policy specification language should be expressive enough to allow for the declaration of meaningful regulatory guidelines. A central difficulty in designing this language is to find a sweet spot between (i) expressiveness power, (ii) readability, and (iii) evaluation performance. Given the extensiveness of GDPR, for example, one temptation is to design a highly-expressive and versatile language that can cover all the corner cases of the law. However, this approach normally leads to complex policies, which tend to be difficult to read and interpret by humans (and thus prone to error), and can incur considerable performance overheads [81]. To strike a balance between these trade-offs, we explore a simple, yet flexible-enough way to express policies that can address some of the most important use-case scenarios w.r.t. GDPR compliance. In cases where it is not necessary to enforce data usage policies such as GDPR or CCPA, policy compliance is still necessary. Clients usually verify the authenticity of nodes using attestation protocols, where each node could have its own attestation mechanism. A policy language allows a client to concisely specify the characteristics of the nodes (in terms of firmware and location) and create an execution environment for the processing of queries (see § 4.3). Our proposed solution is a *declarative policy specification language* (§ 4.3) that is expressive enough to satisfy real-world deployment constraints.

4 Design

We next present the detailed design of IronSafe based on addressing the aforementioned three design challenges (§3.4).

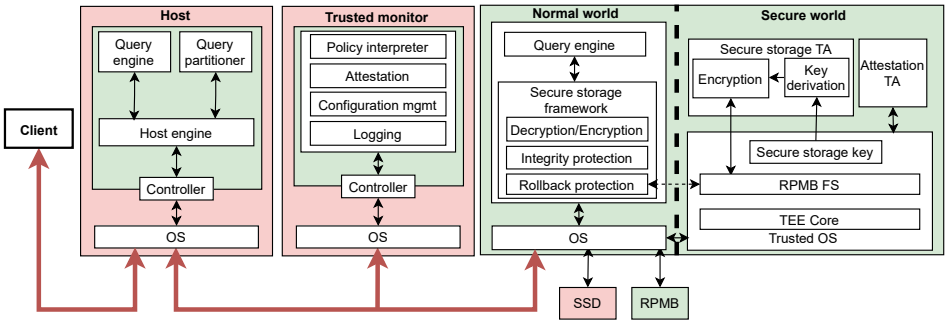


Figure 3: Internals of IronSafe, highlighting the heterogeneous confidential computing framework and trusted monitor

4.1 Heterogeneous Confidential Computing Framework

To address challenge #1, we present the design of a heterogeneous confidential computing framework that allows to securely process SQL queries across the host and storage. Figure 3 presents its components. It consists of a distributed execution runtime that enables secure data processing queries to run across heterogeneous TEE technologies – Intel SGX or ARM TrustZone. The runtime provides a unified interface to an application that intends to take advantage of the CSA for performance speedups. Next, we present the core security mechanisms of our framework intended to protect in-memory query execution and persistent data on untrusted storage medium.

Protection for in-memory query execution. In the host, which we assume to be an x86 server, the host engine runs inside a memory-protected domain implemented by a single SGX enclave. Together with two auxiliary components—the query engine and the query partitioner—it is responsible for handling, partitioning, and dispatching query requests. These components execute within an enclave, untrusted parties cannot inspect their memory or tamper with critical data and operations (e.g., query partitioning).

In contrast to the host, the storage system is based on ARM architecture where the only available TEE-enabling technology is TrustZone (ARM v9 Realms [17] is not released yet). With a TrustZone-based TEE, a strawman approach for securing the storage engine would be to implement as a trusted application (TA) and run it inside the secure world. However, to enable split queries to access persistent data on the storage medium, the storage engine requires full-blown OS support with all the necessary device drivers and file systems. Typically, no such support is available in the secure world given that the trusted OS responsible for managing TAs is bare minimal.

To overcome this problem, we adopt an architecture spanning across both worlds. The secure world only runs certain security-critical functions, including generation of remote attestation quotes, secure storage primitives, and securely bootstrapping the system. These functions are implemented as trusted applications (see Figure 3). The normal world houses a supporting OS stack along with the storage engine's query engine and secure storage system. To propagate trust from the secure world to the normal world, at boot time, the trusted OS performs a hash-based integrity measurement of the normal world software and hands over the control to the normal world OS. Unless the hashes reflect IronSafe's trusted software stack for the normal world the trusted monitor will consider the storage system unsafe and thus ineligible for handling query offloading requests.

As a result, the memory protection of the storage engine is assisted by two mechanisms: i) the storage system's TAs are protected by the secure world's trusted OS, and ii) the storage system's components that run in the normal world are supervised by a trusted OS stack. The firmware is also trusted to provide isolation between offloaded queries, and prevent them from reading and tampering with each other's code and data. Hence, we can guarantee the confidentiality and integrity of offloaded queries and processed data.

Protection for on-storage data. To guarantee the confidentiality, integrity, and freshness of data in the untrusted storage medium, IronSafe relies on several components in the secure storage system (see Figure 3). Some components live in the normal world, with the secure storage TA running in the secure world. It allows the query engine to read or write units of data (fixed at 4 KiB) stored on untrusted medium. Data read or write operations can only be authorized (and thus executed) as long as the trusted monitor has i) vouched for the authenticity of the storage system, and ii) verified the compliance of this setup with the security policy originally provisioned through the client.

The secure storage framework stores persistent state on both untrusted medium (e.g., SSD) and RPMB-backed trusted medium. On an untrusted medium, it reserves a data region for storing the (encrypted) data units sequentially and a meta-data region that preserves a streamlined Merkle tree for integrity protection. For simplicity, IronSafe uses a single secret (symmetric) key to encrypt all the data units, but other management schemes can be adopted (e.g., one key per unit). The secure storage TA generates this key during the system initialization and shares it only with a trusted implementation of IronSafe's storage system running in normal world. To survive system reboots, the encryption key is stored in RPMB memory. For integrity protection, IronSafe generates an HMAC for each data unit. It then recursively builds a Merkle tree also employing HMACs to create the internal nodes and root of the tree. This tree ensures that the data units cannot be silently displaced or suppressed by an adversary with physical access to the untrusted medium.

To prevent rollback attacks and ensure that the Merkle tree itself is fresh, we need to only ensure that the root of the tree is fresh. To this end, we rely on the secure storage TA and on the RPMB region present in an eMMC. First, the storage TA generates a new key derived from a unique, device-specific hardware key, which bounds the data to the CPU. The storage TA uses the new key to generate a HMAC of the Merkle tree root and writes down this HMAC to the RPMB. Freshness is preserved by verifying that the HMAC of the root of the Merkel tree matches the version stored on RPMB.

4.2 Policy Compliance Monitor

To address challenge #2, we present the design of a policy compliance infrastructure that enforces security policies in dynamic,

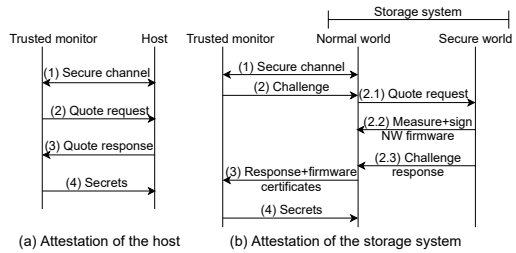


Figure 4: Design of the attestation protocol

heterogeneous cloud environments. The central component of this infrastructure is the trusted monitor, which abstracts away the process of remote attestation of host and storage nodes by acting as their root of trust. Clients are only required to directly trust the monitor, and submit queries and associated execution and data policies.

The trusted monitor service runs inside an SGX enclave, either on the host or on a dedicated server (see Figure 3). To be able to enforce client policies, the first essential step is to obtain hard proof about the configuration of the host and storage system nodes. This proof must ascertain the authenticity and integrity of the software implementing the host engine and in the storage engine. Along with the trusted monitor software they (host engine and storage engine) form IronSafe's TCB and therefore they cannot be tampered with. To obtain such a proof, the trusted monitor runs independent remote attestation protocols with host and storage nodes.

Attestation of the host. Figure 4.a describes the steps during which the trusted monitor remotely attests the integrity and authenticity of the software running in the host engine enclave. After establishing a secure channel over TLS (step 1), the trusted monitor issues a quote request to the host (step 2), who replies with a quote response (step 3). Then, by checking the quote signature, the trusted monitor decides whether the configuration of the host is trustworthy. If so, it generates a public key pair where the public key is certified by the trusted monitor, and sends this information to the host over the secure channel (step 4). The public key certificate will be used by the host to prove to the client that the host has been remotely attested. If the quote verification fails, then the operation is aborted and the host is not authorized to serve IronSafe client requests.

Attestation of the storage system. This operation involves the interaction between the trusted monitor and the attestation TA running in secure world (see Figure 3). As mentioned in § 4.1, we rely on a secure boot process to check the storage engine software. The root of trust for this process is a device-specific, unique key called the *root of trust public key* (ROTPK), and the initial boot firmware is present in tamper proof ROM. This mechanism ensures that only integrity-protected firmware, signed by trusted parties, will run. The attestation TA leverages this mechanism during the remote attestation protocol. Figure 4.b describes the steps of this protocol. The trusted monitor in step (1) requests the certificates from the storage node and challenges the node to prove its authenticity. In step (2) the TA obtains the measurement of the software running in normal world, and step (3) generates a response to the challenge by signing the challenge with a unique key, derived from the ROTPK. In step (4) the TA sends the challenge response, the normal world firmware hash, and the certificate chain generated during secure boot of the storage node. The trusted monitor then verifies the response to the challenge to determine the authenticity of the storage system. The certificate chain

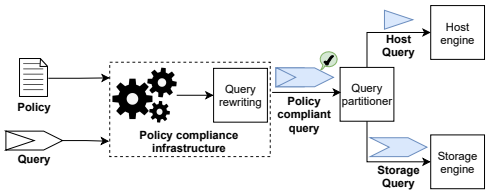


Figure 5: Life of a query in IronSafe

is also verified, and on success, the storage node configuration is derived from the certificate chain. Both the configuration data and the certificate chain are used for policy enforcement as explained next.

Policy-compliant query partitioning. To execute a query while complying with a user-defined security policy, the client first connects to and attests the host. Once authenticated (by presenting a TLS certificate), a mutually encrypted channel is created to exchange data and commands between the actor and IronSafe. The host then submits the query and associated execution policy to the trusted monitor service as shown in Figure 5. The policy compliance infrastructure from Figure 5 shows the host engine working in collaboration with the trusted monitor to ensure policy compliant query processing. The host forwards the query and identity of the connecting client to the monitor, which then checks which of the host and storage nodes comply with the execution policy. The trusted monitor then checks if the connected actor has permissions to read and write data by executing queries. If successful, it sends the list of compliant storage nodes to a compliant host node, along with a session key that allows the host to connect securely to the storage node and process the query as explained in the sections above. The trusted monitor rewrites the client query to be policy compliant based on the results of the policy checks. The host decides whether the query is processed across both the host and storage nodes, or only on the host node or not processed at all. For the first case, it needs to partition the query into portions that run on the host and storage nodes respectively, as shown in Figure 5. If none of the storage nodes comply with the client's execution policy then the entire query may be processed on the host node itself. Once the client request has been completed, the monitor initiates a session cleanup protocol. This deletes any sensitive information that is generated during query processing on the host and storage node, such as temporary tables that are generated by the query engine.

Key management. Once the trusted monitor verifies the authenticity of the hardware and software running on the host and storage nodes, it provides key management services to the nodes. Specifically, it manages and distributes the session keys between the host and storage nodes that is used to create a secure channel for exchanging queries and data between them. After query processing, the trusted monitor revokes the session key and initiates the session cleanup.

Proofs of integrity and authenticity. To provide per query proofs of integrity and authenticity to the client, the trusted monitor signs the execution environment requirements, as specified in the client's execution policy, using its private key provided all nodes in the setup used for query execution satisfy the requirements. The client verifies the signed response with the public key of the trusted monitor.

Separation between the host engine and trusted monitor. The trusted monitor and the host engine run in separate, hardware protected memory regions. More specifically, the trusted monitor is a separate entity for scalability, performance and architectural independence reasons. However, in practice, the trusted monitor can be

run on the same node as the host engine, but inside a separate enclave. We allow the client to connect to the host, instead of the trusted monitor, to differentiate between operations in the control and data paths. The client uses the trusted monitor, indirectly via the host, for control path operations like attestation and policy compliance. It then connects to the host, in the data path, to exchange queries and data. By doing so, we clearly divide the functionality between the host and trusted monitor, and avoid overloading the trusted monitor, thereby ensuring better scalability and performance.

4.3 Declarative Policy Specification Language

To address challenge #3, we present a declarative policy language that can be used to specify data access and execution policies concisely.

Policy language predicates. IronSafe's policy language allows a client to concisely specify the execution context associated with the execution of a query in a split manner, across the host and storage. A policy in IronSafe is a combination of predicates described in Table 1. A policy evaluates to true, if and only if all the predicates specified in the policy evaluate to true. In IronSafe we have two kinds of policies: *data access* and *data execution* policies. Data execution policies are specified by the client when executing queries. They can be any combination of location and firmware predicates. Data access policies are created (or updated) by the client that creates the database and tables in that database. These policies are associated with permissions for reading or writing on the database or its tables. They are of the form *perm :condition.* The condition is a combination of predicates and evaluates to true only if all predicates evaluate to true. The predicates are joined using either a '|', which evaluates to true if any of the predicates are true, or a '&', which evaluates to true if and only if both predicates evaluate to true. An example data access policy is shown below in which user A can read and user B can write to the database.

$$read :- sessionKeyIs(K_A)$$
$$write :- sessionKeyIs(K_B)$$
$$exec :- fwVersionStorage(latest) \& fwVersionHost(latest)$$

More complex policies, that can be used in the real world to adhere to GDPR rules are described below. The table describes four kinds of predicates that can be used to compose policies. *sessionKeyIs* enables a client to derive policies to restrict access, by reading or writing, to data, only to certain users based on their identity key K_x. The second kind of predicates enable a client to run data processing operations on nodes, host or storage, only if they are located in a region or regions. The client uses the *storageLocIs* predicate to enforce the offloading of queries only to storage nodes located in region l. If no nodes satisfy this property then, as described before, the entire query may be run on the host node itself. The client uses the *hostLocIs* predicate to enforce the running of queries only on host nodes that are located in a specific region l. Similarly, the client uses the firmware predicates to enforce processing of queries on nodes that are running software greater than or equal to a certain version v. The final two predicates enable a client to specify how their data can be used by an external entity and record all operations that are performed by that entity.

GDPR anti-pattern use-cases. To show the effectiveness of our language, we describe GDPR anti-patterns use-cases [109, 116].

(1) *Timely deletion of data:* GDPR allows data owners to force their data to be deleted after a certain period of time. However, fast deletion is a hard problem to solve due to data replication across various storage systems [55]. Hence, IronSafe's policy language allows provisioning a timestamp, after which data cannot be accessed. As shown in the data access policy below, only actors A and B, represented by their respective identity keys, may access the data in the database. Also, they can access records only if the access time (T) is before the expiry time (TIMESTAMP) of the record. To enforce this, the trusted monitor performs two tasks. First, during data creation it rewrites the insert queries to include an extra column that represents the expiry time of the record. Secondly, during query processing it rewrites the query to filter out the expired records i.e., whose expiry time (TIMESTAMP) is lesser than the access time (T).

Predicate	Meaning
Relational predicates	
eq(x,y), *le*(x,y), *lt*(x,y)	x = y, x ≤ y, x < y
ge(x,y), *gt*(x,y)	x ≥ y, x > y
Session predicates	
sessionKeyIs(K)	Client is authenticated with key K
Location predicates	
storageLocIs(l)	storage device is located in location l
hostLocIs(l)	host is located in location l
Firmware predicates	
fwVersionStorage(v)	storage device is running software version v
fwVersionHost(v)	host is running software version v
Data predicates	
logUpdate(l, c)	Appends log l with content c
reuseMap(bm)	Bitmap bm indicating services with whom data can be shared

Table 1: IronSafe policy language predicates

$$\textbf{read}: -sessionKeyIs(K_A) \mid sessionKeyIs(K_B) \,\&\, le(T, TIMESTAMP)$$

(2) *Prevent indiscriminate use of data:* GDPR specifies that personal data collected for a specific purpose cannot be used for anything else, such as to avoid past violations [53]. To allow this, we introduce a new predicate, called *reuseMap*, that allows for opting-in/out to letting certain data being used for each service individually, as shown in the example below. The *reuseMap* predicate stores a bitmap m that represents the list of services allowed to access the data. The trusted monitor performs two tasks. First, during insertion of data, it rewrites the insertion query to include a new column that represents the reuse map. Secondly, during query processing it converts the connecting client's identity into a bitmap, by referring to an internal database containing mappings between identity keys and positions in the bitmap. It then rewrites the query to filter out records that do not want to be included in the processing of the request.

$$\textbf{read}: -reuseMap(m)$$

(3) *Transparent sharing and reselling of data:* GDPR specifies that data owners have a right to obtain a copy of all personal data that a controller has collected and has shared with an external party. To enforce this, IronSafe's policy language provides a logging predicate that can be specified on data creation, along with the format of the data to be logged. For example, in the policy shown below, the data producer may request that the identity key K and the query Q submitted by a data consumer, be logged. The trusted monitor is responsible for logging this information into log l. At a later point in time, it is possible to audit whom the data has been shared with by requesting the trusted monitor for the logged data.

$$\textbf{read}: -logUpdate(l,(K,Q))$$

(4) *Risk agnostic data processing:* Article 5 of GDPR mandates that personal data should be processed in a fair, lawful and transparent manner. The controller is responsible for enforcing these requirements and demonstrate compliance with all the features. We design our system to handle secure processing of external queries. To provide an extra layer of assurances and combat this anti-pattern, the data producer can combine access policies with access logging. A data consumer, who has restricted access to the data, can further specify an execution environment for processing queries using the *exec* operation as shown below. In the example policy shown below, query processing can be performed on the data only by actors A and B. Moreover, client A or B can specify the execution environment for processing of queries with the help of location and firmware predicates. Requests will also be logged by the trusted monitor.

read: $-sessionKeyIs(K_A) \mid sessionKeyIs(K_B)\ \&\ logUpdate(l,(K,Q))$
exec: $-storageLocIs(\text{"uk"}) \mid storageLocIs(\text{"us"})\ \&$
$fwVersionHost(\text{"latest"})\ \&\ fwVersionStorage(\text{"latest"})$

(5) *Data breaches:* Article 5 from GDPR requires that the controller should notify the data owner of any data breaches within 72 hours, after becoming aware of it. To enforce this property, one can specify that all requests be logged to a secure, confidentiality and integrity protected log. The trusted monitor is trusted to operate correctly and only allows clients with correct permissions to access data. Since we trust the underlying hardware and rely on the integrity of the hardware-assisted enclave, we assume that the software running inside the enclave, query engine and trusted monitor, is protected. Hence, the logging mechanism cannot be circumvented.

read: $-sessionKeyIs(K_A)\ \&\ logUpdate(l,(K,Q))$

Enforcement of execution environment. The policy language is also used by the client to describe and enforce the execution environment for query processing. This is useful in cases where it is not necessary to enforce data protection and usage policies such as GDPR, but it is required to enforce processing of the query in a secure environment. A client, for example, specifies that the query must be processed on nodes that are located in a particular region and running the most up-to-date firmware, as shown below.

exec: $-fwVersionHost(\text{"latest"})\ \&\ fwVersionStorage(\text{"latest"})\ \&$
$storageLocIs(\text{"uk"}) \mid storageLocIs(\text{"us"})$

Attribute-based access control. Our example use-cases consider access control methods that restrict access to the entire database. However, our query rewriting mechanism also supports attribute-based access control. In fact, our rewriting mechanism is compatible with previous work [81] that enforces attribute-based policies based on a combination of the individual columns in the database.

5 Implementation

We next describe a proof-of-concept implementation of IronSafe.

Query processing system. To showcase the effectiveness of our approach, we built a CSA-aware query processing system for secure SQL query processing. The database portion of our system is based on SQLCipher [114], which is based on SQLite (v 3.31.0) [113]. We establish two SQLite instances, on the host and on a storage server. The x86 host query processing system includes an in-memory SQLite instance, that is kept entirely in the host enclave memory. This in-memory database operates on a single table, containing records that are filtered out by the storage system. The storage query processing system runs an on-disk SQLite instance, which executes the offloaded query (e.g., filtering of records) on the database residing in the untrusted persistent storage and ships the output (e.g., filtered out records) to the in-memory instance running on the host. The host query processing system is responsible for splitting an SQL query into queries that run on the host and the storage system. On the storage system side queries include filters to remove unwanted records, while the host side queries perform further operations, including group-bys and aggregations, on the filtered out records.

Heterogeneous confidential computing framework. Regarding protection of volatile state, on the host we use SCONE [22] for shielding in-memory processing of queries on x86. The trusted monitor, and the host engine runs inside SGX enclaves. SCONE is run in its default mode with 4 syscall queues and 250 entries per queue. However, we increase the heap size to its maximum addressable size, 4 GiB, which is limited by the SGX hardware. On the storage system, we use OP-TEE [92] version 3.4, a trusted OS for ARM TrustZone, to manage the secure world, alongside ARM Trusted Firmware (ATF) [19]. These software components along with a few trusted applications implement the key functionality of secure boot, remote attestation, and secure storage. The normal world is used for the CSA runtime and on-disk SQLite instance. Thereby, offloaded queries are processed in the normal world itself after secure boot. Host and storage system communicate via a secure TCP connection.

Regarding on-storage data protection, our current implementation of the secure storage system is tightly coupled with SQLite's VFS and page layers. Specifically, it is dependant on the SQLite codec API to insert a callback at the paging layer, which is invoked whenever a page is read and written from/to the storage system. We rely on SQLCipher [114], which depends on OpenSSL version 3.0.0, to encrypt each page (4 KiB) in the database individually, using the 256-bit AES encryption algorithm in the CBC mode. Each page also stores a random initialization vector (IV) and a message authentication code (HMAC-SHA512), which is derived from the encrypted page data and the random IV. We use a Merkle tree containing the hashes of all pages in the database. We leverage the storage TA to write to the RPMB region of the eMMC. It uses a 128 bit TA storage key (TASK), derived from the hardware unique key, present on the device.

Trusted monitor. The trusted monitor provides a unified interface for attestation, key management, and policy compliance. The trusted monitor internally handles the two heterogeneous (x86 and ARM) domains via different interfaces. For the host (x86), we leverage SCONE's configuration and attestation service using which the client can verify the authenticity of the hardware and the software running inside an enclave on the host. For the storage system we rely on a trusted application running in the secure world to generate the integrity signature of the firmware running in the normal world.

Before delivering a query, the trusted monitor runs a policy interpreter that is responsible for interpreting client execution policies and enforcing owner's access policies. The policy interpreter is implemented entirely in python. The policy is parsed into a python dictionary that is then used by the monitor to authenticate requests and enforce data owner defined access policies.

Networking layer. To secure communication between the monitor and the storage (server or device) along the untrusted channel, we implement a trusted networking layer. Depending on the deployment

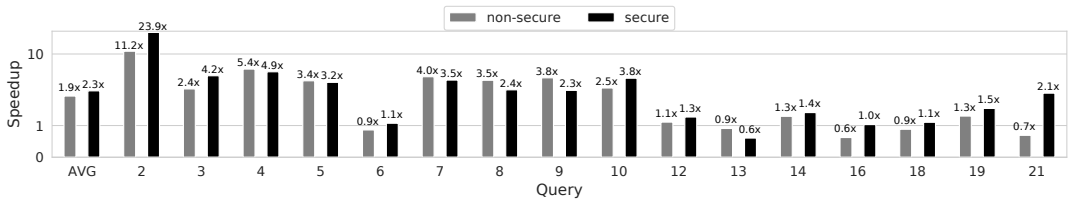


Figure 6: TPC-H queries execution time speedup due to CS execution while running without any security environment – non-secure (*hons* vs *vcs*), and within a secure environment – secure (*hos*, vs *scs*). Higher is better.

Abbrv.	System	Split execution
hons	Host-only-non-secure	No
hos	Host-only-secure	No
vcs	Vanilla-CS	Yes
scs	IronSafe	Yes
sos	Storage-only-secure	No

Table 2: System configurations and their naming schemes

model, the layer can be configured as: NVMe/PCIe, NVMe over fabrics (NVMe-oF), or a TCP. For our evaluation, we use TLS over TCP/IP. The TLS session is created each time a new client request is made, and is torn down once the client request is complete. A new session key is used for each session. The monitor and storage system participate in a symmetric key exchange process, to create an encrypted TLS channel. The sender is responsible for serializing records and the receiver deserializes these records to be added to the in-memory table on the host.

6 Evaluation

6.1 Experimental Setup

Since there is no commercially available CSA platform supporting fully programmable hardware stack, we build our own CS hardware infrastructure that includes an SGX-enabled x86 host connected to a TrustZone-enabled ARM storage server. Specifically, the host features an Intel CPU with SGX, i.e., Intel Core i9-10900K CPU with 10 cores at 3.7GHz (caches: 32 KiB L1; 256 KiB L2; 20 MiB L3), and 64 GiB of RAM. The I/O devices on the host include an Intel XL710 Ethernet controller dual 40 GbE. The ARM-based storage server is the Solidrun Clearfog CX LX2K board [43], which supports ARM TrustZone thanks to the NXP Layerscape LX2160A SoC, a 16-core ARM Cortex A72 at 2.2GHz (caches: 32 KiB L1; 8 MiB L2), and 32 GiB of RAM. The I/O devices include a 40 GbE network interface, split into 4x 10 GbE ports at the physical-level, and a Samsung 970 EVO Plus 1 TB NVMe drive (sequential reads up to 3329 MB/s measured with `fio`). The host and storage server are connected via a 40 GbE switch. The host OS is NixOS (kernel 5.11.21), while the storage server runs a patched version of the Linux kernel, version 5.4.3, with the same Ubuntu 18.04 —Arm distribution. The storage server additionally runs the OP-TEE secure OS version 3.4 in the ARM secure-world.

We evaluate our system using 16 out of 22 SQL queries from the TPC-H benchmark suite [120]. This is because even if queries are automatically partitioned, the resulting split queries are not suitable for offloading, similarly to [37, 60, 71].

We run all benchmarks in the five configurations described in Table 2. The table describes a total of three baseline configurations: *hons* and *hos*, which run the entire software on the host machine while connecting via NFS to the storage server, and *sos*, which runs entirely on the storage node (with direct attached storage). Note that for fair comparison we choose NFS among other NAS or SAN technologies because we measured the same single-thread network bandwidth (850MB/s) achievable with our IronSafe. The other two configurations run queries among both the host and the storage server (*vcs*, *scs*), using IronSafe without security and with security enabled. For every benchmark in each configuration we run 10 experiments, we report the average of the execution time.

6.2 End-to-End System Performance

Methodology. To assess the performance improvements brought by the NDP execution as well as the possible overheads introduced by IronSafe we focus on the following experimental configurations: *hons*, *hos*, *vcs*, *scs*. Specifically, we measure each query execution time in every configuration, as well as how much data is transferred between host and storage servers, and the cost of IronSafe's component.
Results. Figure 6 shows the relative end-to-end performance of running TPC-H queries in both the non-secure and secure cases. Similarly to previous works [37, 60, 71], we realized that not all queries take advantage from vanilla CS processing. Although we observed up to 11.2× speedup when running a query with vanilla CS versus host only (*hons*/*vcs*), few queries (#6, #13, #16, #18 and #21), run slower with CS. When switching to secure execution (*hos*/*scs*), a similar trend is observed, up to 23.9× faster, with an average of speed of 2.3×. In this case #13 shows a slowdown during secure execution. Next, we analyze our results.
Data movements. Figure 7 shows the reduction in the data exchanged between host and storage server when using CSA, this is calculated as the ratio of the number of pages processed in host only versus the computational storage– the same graph applies for the secure and non-secure case. We observe that query speedup is almost directly correlated with the IO reduction for both secure and non-secure cases. However, this doesn't apply to query #21 because manual partitioning produces a computationally intensive query, which is not suitable to run on the storage CPU.

Reduced data movements are advantageous in IronSafe, as the host side application has to exit and enter the enclave fewer times to fetch data for processing. Entering into and exiting from the enclave are costly operations. This is the case of query #21 that shows a slowdown with vanilla CS but a great speedup with IronSafe: the cost of exiting and entering the enclave is more than the cost of offloading the compute intensive query to the storage server.
Security overheads. Figure 8 shows the relative cost breakdown of running each selected TPC-H query with IronSafe. The "ndp" component in the Figure is equivalent to the cost of Vanilla CS (*vcs*)— i.e., the non-secure version of CSA. "Other" overhead includes the cost for channel encryption and instantiation of storage side CS service, which are negligible. We note that most of the overhead

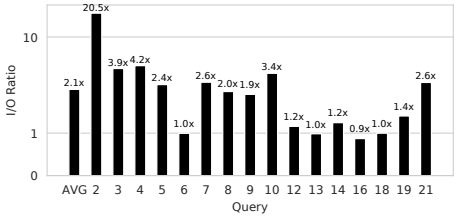


Figure 7: I/O data reduction when using CSA (*vcs* or *scs*). Ratio of bytes transmitted over the network without CS and with CS. Higher value corresponds to higher reduction.

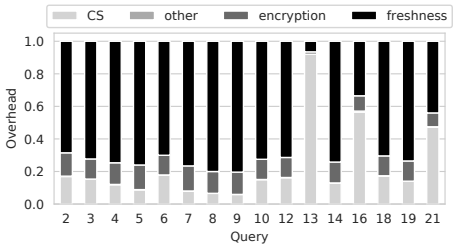
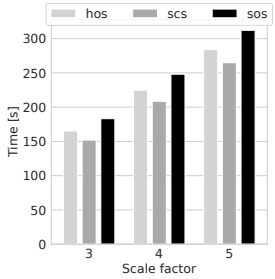
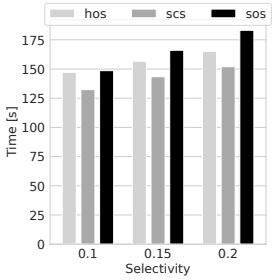


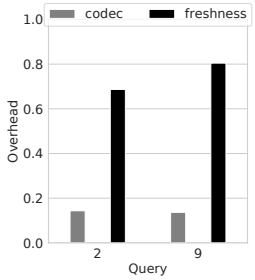
Figure 8: Relative cost breakdown of each TPC-H queries running with IronSafe. CS corresponds to *vcs*, other includes all costs other than encryption and freshness (Merkle tree).



(a) Impact of data size on query processing time on filter of selectivity 0.2.



(b) Impact of selectivity of filter on query processing time on data of scale factor 3.



(c) Impact of secure storage framework.

Figure 9: Heterogeneous TEE and secure storage overheads.

comes from guaranteeing the freshness of pages read from untrusted storage. Data transfer of filtered records takes no extra time, as data is transferred asynchronously between the storage server and host. Hence, despite the high overheads, mostly due to maintaining data freshness, IronSafe still performs better than when the application is run purely on the host in a secure manner.

6.3 Heterogeneous Confidential Computing Framework

To assess the performance impact of running queries using our confidential computing framework, we use three configurations: *hos*, *scs*, and *sos*, with different input size (databases) and selectivity (filter). For each of the three configurations, we focus on query 1 from TPC-H whose filter was tweaked to change selectivity.

- *Input size:* Figure 9a shows the impact of processing a query, with a single filter predicate, by varying the data size while keeping the filter selectivity constant.
- *Query selectivity:* Figure 9b shows the impact on processing a query, with a single filter predicate, by varying the filter selectivity from 10 % to 20 % with a constant data size at a scale factor of 3.

As expected, Figures 9a and 9b, show that the performance of IronSafe (*scs*) is better than *hos* and *sos* in all cases—in these graphs, lower is better. For *hos* running in the secure mode, performance drops due to multiple enclave exit/enter to fetch data from disk and EPC paging caused due to the limited size of the enclave memory in Intel SGX, which equals 96 MiB in our setup. The space is taken up by the Merkle tree required to maintain freshness of data read in from storage medium. Data with scale factors (i.e database size) 3, 4, and 5 take up 59 MiB, 78 MiB and 98 MiB of the enclave memory. This causes EPC paging and reduces performance, which is depicted clearly in Figure 9a. Whereas, in the *sos* configuration, performance drops because we need to run complex query operations such as group-by and aggregations on the storage server itself—with a weaker CPU.

These problems do not arise in IronSafe because of three reasons. First, the Merkle tree is used to verify the data on the storage system itself; this is not limited by the memory size. Secondly, data is read into the host TEE, after filtering them out, into a single table in memory. Pages belonging to the table are accessed in a sequential manner, as the host portion of the query performs a full table scan. Moreover, the number of pages in memory on the host, on which the query operates, is less compared to the pure host case, due to the initial filtering out of data. Hence, the effect of EPC paging is minimal in IronSafe. Finally, the host in IronSafe is responsible for running the most compute intensive parts of a query.

6.4 Constrained Resource Evaluation at CSA

To evaluate the constrained resources on the CSA side, we perform additional experiments across the following dimensions using TPC-H queries: (a) compute cores, (b) memory, (c) storage engine scalability.

(a) Limited compute cores. To understand the effectiveness of IronSafe when run across a storage server with less compute cores, we hotplug CPUs on the storage server and run the TPC-H queries for the following configurations: *hos*, *scs*. We vary the number of CPUs on the storage server with 1, 2, 4, 8 and 16 CPUs. We then measure the end-to-end query execution time for both the configurations. Figure 10 shows the relative end-to-end performance of running TPC-H queries in the secure case(*hos* and *scs*) against an increasing number of CPUs. As shown in the figure, the relative performance generally improves with increasing number of CPUs on the storage server. Multiple queries (#2, #3, #4, #5, #7, #10) show speedups even when there is only 1 CPU available on the storage server as the offloaded query is not very computationally intensive, i.e., it completes in a

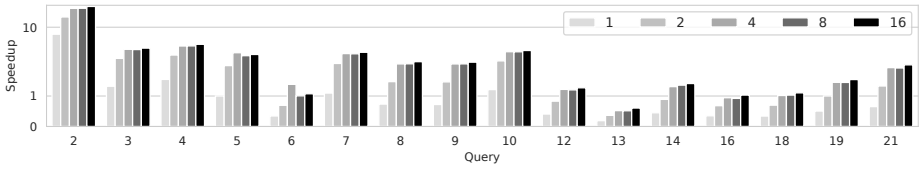


Figure 10: TPC-H queries execution speedup due to increasing number of CPUs on the storage server– in a secure environment – secure (*hos*, vs *scs*). Higher is better.

short period of time compared to the *hos* configuration. Generally, as the number of cpus increases from 1 to 4, the performance of the query execution also increases. This is because the application on the storage server that is running the offloaded queries is multi-threaded, and contention for CPU cores between the application threads and OS services (network and filesystem) reduces as the number of CPU cores is increased. As expected, query #13 does not perform well as its offloaded portion does not reduce data movements.

(b) Restricted physical memory. To understand the effectiveness of IronSafe when run across a storage server with restricted physical memory, we vary the amount of memory that can be utilised by the query engine on the storage server from 128 MiB to 2 GiB using `cgroups` [40]. We then measure the query execution time of offloaded TPC-H queries. Figure 11 shows the speedups of each TPC-H query normalised w.r.t. to query execution on a storage side application running with only 128 MiB of memory on the storage server. We see that many offloaded queries (#2, #4, #6, #12, #16, #18) are not memory intensive and can be run within only 128 MiB of memory. Other queries (#3, #5, #7, #8, #9, #10, #14, #19, #21) speedup when the memory available to the storage side application increases to 256 MiB with no further improvements when the available memory is increased to 2 GiB. Query #13 offloaded portion performs a memory intensive join; hence, the performance improves as more memory is available.

(c) Storage side scalability. To assess the query engine scalability on the storage server, we run multiple instances of SQLite in different threads, each operating on its own copy of an encrypted, integrity and freshness protected database of scale factor 3. We vary the number of instances as: 1, 2, 4, 8, 16. We then measure the offloaded query's execution time of each instance for TPC-H queries. Figure 12 shows the cumulative execution times of all queries across all instances, normalized to a single instance. All queries, except for query 13, scale linearly with increasing number of instances.

6.5 Secure Storage

To measure the storage system's security overheads, we run queries entirely on the storage server (*sos* configuration). For brevity, we only show the results for query #2 and #9 in Figure 9c. Query #2 and #9 spend about 70% and 80%, respectively, of their total time verifying freshness of database pages and about 15% of total time for decryption of these pages.

The overheads observed are due to the implementation of secure storage primitives. The responsibility of maintaining confidentiality, integrity and freshness lies with the database engine running on the storage server. The database engine decrypts a page each time a page is read from the untrusted storage. It also checks whether that page is fresh by traversing the Merkle tree for each read request. Query #9 and #2 request pages for processing, approximately 23M and 200K times respectively, which account for the high overheads.

Component	Breakdown	Time(ms)
Host	CAS response	140
Storage server	TEE	453
	REE	54
	Interconnect	42
Total		689

Table 3: Host and storage system attestation breakdown.

GDPR Anti-pattern	Non-secure	IronSafe	Overhead
#1: Timely deletion	2.3 ms	12.8 ms	5.6×
#2: Indiscrimination	1.9 ms	14.8 ms	7.8×
#3: Transparency	3.5 ms	16 ms	4.6×
#4: Risk agnostic	7.2 ms	34.9 ms	4.8×
#5: Data breaches	7.1 ms	38.1 ms	5.4×

Table 4: Policy evaluation overheads for GDPR anti-patterns.

6.6 Policy Compliance Monitor

We consider two scenarios to explain the effectiveness of the trusted monitor (§ 4.2). In the first scenario, the client has to individually attest the host and storage node before submitting queries for execution. In the second, the client attests only the host, running the attestation service, and submits queries as described in section 4.2. In both scenarios the client executes on the same node as the trusted monitor.

Table 3 shows that when the client attests both the host and the storage server, it has to spend approximately 690ms, before it can submit requests to the host. However, using the attestation protocol described in Section 4.2, by using the trusted monitor to attest the storage node, the client only needs to authenticate the host, and can submit a request in 80% less time (140 ms).

6.7 GDPR Anti-Pattern Use-Cases

To gauge the overheads of policy enforcement, we evaluate the GDPR use cases described in § 4.3. We run the policy interpreter inside the SGX enclave using SCONE. The log file is stored encrypted and integrity protected, on untrusted storage, using SCONE fileshield, which provides transparent file encryption and decryption.

Table 4 shows the absolute execution times of all use cases when run in a secure and non-secure manner. The table also shows the relative overheads between the two setups. Use cases #1, #2 and #3 are relatively faster compared to others, as they only interpret policies and check them against access and execution constraints. However, use cases #4 and #5, in addition to interpreting and checking policies, update the log with the query, identity and time it was issued. The log's integrity and confidentiality is protected by SCONE fileshield.

6.8 Summary of Results

IronSafe enables a 2.3× speedup on average for query execution of most TPC-H queries, while also ensuring that the confidentiality, integrity and freshness of query execution is preserved. It does so by

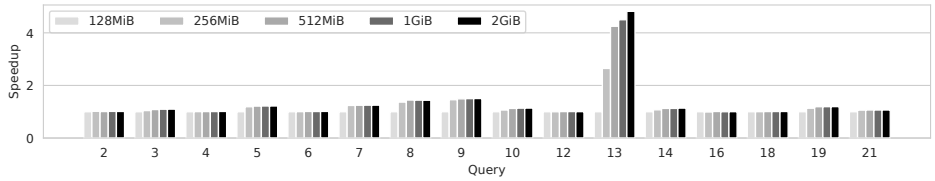


Figure 11: TPC-H queries execution time normalized wrt to 128 MiB **of memory for query processing on the storage server.**

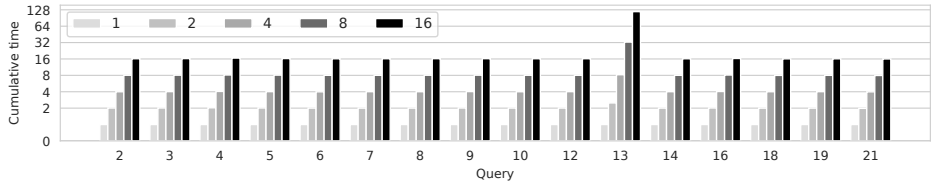


Figure 12: TPC-H queries cumulative execution time of all instances normalized wrt to a single *sqlite* instance on the storage server when run within a secure environment.

ensuring that the reduction in IO between the host and storage node is reduced by 2.1× on average, while incurring reasonable overheads as shown in Figure 8. Additionally, we also show that IronSafe can be run, with speedup in query execution, across storage nodes with less compute and memory as shown in Figures 10 and 11 respectively.

7 Related Work

Confidential computing. We mainly focus on ARM TrustZone and Intel SGX TEEs for confidential computing [14, 25, 57, 64]. TrustZone is used in several workflows [95], including monitoring kernel integrity [23], maintaining secrets [129], managing cryptographic material [79], trusted language runtimes [105], or remote hardware management [34]. Likewise, Intel SGX has been extensively used to support many applications [22, 30, 42], including Web search [83], storage systems [26, 119], distributed systems [121], FaaS [123], networking [97, 119, 122], machine learning [90, 100], and data analytics systems [76, 108, 130]. Similarly, IronSafe enables heterogeneous TEEs to interoperate, such as HETEE [131] and Graviton [127]. In contrast, we present the first heterogeneous confidential computing framework for CSA.

Secure query processing systems. Multiple secure storage and query processing systems have been proposed based on different TEEs, varying security guarantees, and storage interfaces [12, 12, 27, 32, 70, 73]. Secure query processing systems [94, 98, 124] describe approaches to execute queries directly on encrypted data, and propose various encryption schemes to make this possible. EnclaveDB [99] presents the design of an in-memory database running inside an SGX enclave. There exist work in secure query processing on privacy-preserving analytics [49, 72, 80, 86], oblivious query processing [45, 83, 96], secure MPC [126]. In contrast, IronSafe strives to ensure the security of data in transit, at rest, and processing across heterogeneous computational domains, enforcing access policies.

Database access control prevents access from unauthorized parties to the database, and is essential. In [11, 77] data access control is enforced by the data owner, at the granularity of a tuple, by introducing access policies at the data cell-level. Other methods include query rewriting techniques [62, 84], extensions to SQL language [41], use authorization views [103], and query control [110]. Similarly, in IronSafe we let the user specify data access policies which are enforced, later, by rewriting queries to restrict access to data on a per tuple basis.

Policy language and compliance. Several policy languages can be used to restrict access to data and execution of queries [31, 51, 52, 73, 81, 81]. IronSafe presents a new declarative policy language that allows users to describe data access and execution policies. Recent efforts [48, 54, 67, 73–75, 81, 82, 125] enable clients to describe policies in a declarative language, a thin software layer at the object, file or query level ensures policy compliance data access. Further, attestation systems [58, 107] provide for fast and scalable attestation. IronSafe builds upon prior work to create a unified abstraction to attest and policy compliance across the heterogeneous CSA.

8 Conclusion

We present IronSafe, a design approach for building a secure, policy-compliant, and high-performance query processing infrastructure for untrusted clouds. To achieve these goals, our work underpins on three main contributions: (1) a heterogeneous confidential computing framework and associated secure storage; (2) a policy compliance monitor to provide a unified attestation and enforcement interface; and (3) a declarative policy compliance language to concisely express a rich set of polices. We have built the end-to-end query analytics infrastructure that exposes a declarative (SQL) query and associated execution policy interface. Our evaluation using the GDPR anti-patterns and TPC-H SQL benchmark queries shows reasonable overheads, while providing strong security and policy-compliance.

Limitations. While we rely on a simple query partitioning strategy by adapting the MySQL partitioner with simple heuristics, a generic query partitioning framework for CSA is beyond the scope of this work and been actively investigated. As part of the future work, we plan to build on the advancements in databases [13, 60] to design a compiler that automatically partitions queries between the host and storage systems, while considering TEEs' architectural limitations.

Software artifact. We release IronSafe as an open-source project [44].

Acknowledgements. We thank our anonymous reviewers for their helpful comments. This work was supported in parts by a Huawei Research Grant, by Fundação para a Ciência e Tecnologia (FCT) under project UIDB/50021/2020 and grant 2020.05270.BD, via project COSMOS (via the OE with ref. PTDC/EEI-COM/29271/2017), via the "Programa Operacional Regional de Lisboa na sua componente FEDER" with ref. Lisboa-01-0145-FEDER-029271 and an FCT grant with ref. SFRH/BD/146231/2019.

References

[1] 175 zettabytes by 2025. https://www.forbes.com/sites/tomcoughlin/2018/11/27/175-zettabytes-by-2025/.

[2] The 2020 data attack surface report. https://1c7fab3im83f5gqiow2qqs2k-wpengine.netdna - ssl.com / wp - content / uploads / 2020 / 12 / ArcserveDataReport2020.pdf.

[3] Amazon aurora. https://aws.amazon.com/rds/aurora/.

[4] Amd secure encrypted virtualization (sev). https://developer.amd.com/sev/.

[5] Aqua (advanced query accelerator) for amazon redshift. https://aws.amazon.com/redshift/features/aqua/.

[6] Arm Confidential Compute Architecture. https://www.arm.com/why-arm/architecture/security-features/arm-confidential-compute-architecture.

[7] Data of 533 million facebook users being sold via telegram bot: Report. https://www.business-standard.com/article/technology/data-of-533-million-facebook-users-being-sold-via-telegram-bot-report-121012600279_1.html.

[8] Hacker pretends to be evan spiegel to steal snapchat employee data. https://www.forbes.com/sites/thomasbrewster/2016/02/29/snapchat-data-leak/.

[9] Intelligent query processing in sql databases. https://docs.microsoft.com/en-us/sql/relational-databases/performance/intelligent-query-processing?view=sql-server-ver15.

[10] I. F. Adams, J. Keys, and M. P. Mesnier. Respecting the block interface - computational storage using virtual objects. In *Proceedings of the 11th USENIX Conference on Hot Topics in Storage and File Systems (FAST)*, 2019.

[11] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Hippocratic databases. In *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB)*, 2002.

[12] A. Ahmad, K. Kim, M. I. Sarfaraz, and B. Lee. Obliviate: A data oblivious filesystem for intel sgx. In *27th Annual Network and Distributed System Security Symposium (NDSS)*, 2018.

[13] Y. Ahmad, O. Kennedy, C. Koch, and M. Nikolic. Dbtoaster: Higher-order delta processing for dynamic, frequently fresh views. *CoRR*, 2012.

[14] Alibaba Cloud's Next-Generation Security Makes Gartner's Report. https://www.alibabacloud.com/blog/alibaba - clouds - next - generation - security-makes-gartners-report_595367.

[15] S3 select and glacier select – retrieving subsets of objects. https://aws.amazon.com/blogs/aws/s3-glacier-select/.

[16] 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 (HASP)*, 2013.

[17] arm cca. https://developer.arm.com/architectures/architecture - security-features/confidential-computing.

[18] ARM. Computational Storage. https://www.arm.com/solutions/storage/computational-storage, 2020.

[19] Trusted firmware a (tf-a). https://www.trustedfirmware.org/projects/tf-a/.

[20] Arm. building a secure system using trustzone technology. http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf.

[21] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[22] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'Keeffe, M. L. Stillwell, and et al. Scone: Secure linux containers with intel sgx. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI)*, 2016.

[23] A. M. Azab, P. Ning, J. Shah, Q. Chen, R. Bhutkar, G. Ganesh, J. Ma, and W. Shen. Hypervision across worlds: Real-time kernel protection from the arm trustzone secure world. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2014.

[24] Microsoft azure. https://azure.microsoft.com/en-us/.

[25] Azure confidential computing. https://azure.microsoft.com/en-us/solutions/confidential-compute/.

[26] M. Bailleu, D. Giantsidi, V. Gavrielatos, D. L. Quoc, V. Nagarajan, and P. Bhatotia. Avocado: A secure In-Memory distributed storage system. In *2021 USENIX Annual Technical Conference (USENIX ATC)*, 2021.

[27] M. Bailleu, J. Thalheim, P. Bhatotia, C. Fetzer, M. Honda, and K. Vaswani. SPEICHER: Securing lsm-based key-value stores using shielded execution. In *17th USENIX Conference on File and Storage Technologies (FAST)*, 2019.

[28] A. Barbalace, M. Decky, J. Picorel, and P. Bhatotia. BlockNDP: Block-storage Near Data Processing. In *Proceedings of the 1st International Middleware Conference Industrial Track*, 2020.

[29] A. Barbalace, A. Iliopoulos, H. Rauchfuss, and G. Brasche. It's time to think about an operating system for near data processing architectures. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems (HotOS)*, 2017.

[30] A. Baumann, M. Peinado, and G. Hunt. Shielding applications from an untrusted cloud with haven. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2014.

[31] K. Beedkar, J.-A. Quiané-Ruiz, and V. Markl. Compliant geo-distributed query processing. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD)*, 2021.

[32] D. I. Bernard, S. G. Haryadi, J. F. Ariel, and H. Henry. Strongbox: Confidentiality, integrity, and performance using stream ciphers for full drive encryption. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2018.

[33] P. Bhatotia, R. Rodrigues, and A. Verma. Shredder: Gpu-accelerated incremental storage and computation. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies (FAST)*, 2012.

[34] F. Brasser, D. Kim, C. Liebchen, V. Ganapathy, L. Iftode, and A.-R. Sadeghi. Regulating arm trustzone devices in restricted spaces. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2016.

[35] The 56 biggest data breaches. https://www.upguard.com/blog/biggest-data-breaches.

[36] 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)*, 2018.

[37] W. Cao, Y. Liu, Z. Cheng, N. Zheng, W. Li, W. Wu, L. Ouyang, P. Wang, Y. Wang, R. Kuan, Z. Liu, F. Zhu, and T. Zhang. POLARDB meets computational storage: Efficiently support analytical workloads in cloud-native relational database. In *18th USENIX Conference on File and Storage Technologies (FAST)*, 2020.

[38] California consumer privacy act (ccpa). https://oag.ca.gov/privacy/ccpa.

[39] Why ceph on arm is a killer combination for enterprise storage? https://softiron.com/blog/why - ceph - on - arm - is - a - killer - combination - for-enterprise-storage/.

[40] Control Group v2. https://www.kernel.org/doc/Documentation/cgroup-v2.txt.

[41] S. Chaudhuri, T. Dutta, and S. Sudarshan. Fine grained authorization through predicated grants. In *Proceedings of the 23rd International Conference on Data Engineering ICDE*, 2007.

[42] C. che Tsai, D. E. Porter, and M. Vij. Graphene-sgx: A practical library OS for unmodified applications on SGX. In *USENIX Annual Technical Conference (USENIX ATC)*, 2017.

[43] Solidrun clearfog cx lx2k board. https://shop.solid - run.com/product - category/embedded-computers/nxp-family/clearfog-cx-lx2k/.

[44] IronSafe code. https://github.com/harshanavkis/ironsafe.

[45] N. Crooks, M. Burke, E. Cecchetti, S. Harel, R. Agarwal, and L. Alvisi. Obladi: Oblivious serializable transactions in the cloud. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 727–743, Carlsbad, CA, Oct. 2018. USENIX Association.

[46] J. Do, Y.-S. Kee, J. M. Patel, C. Park, K. Park, and D. J. DeWitt. Query processing on smart ssds: Opportunities and challenges. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2013.

[47] J. Do, S. Sengupta, and S. Swanson. Programmable solid-state storage in future cloud datacenters. *Communications of the ACM*, 62(6):54–62, 2019.

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

[49] F. Emekci, D. Agrawal, A. Abbadi, and A. Gulbeden. Privacy preserving query processing using third parties. In *22nd International Conference on Data Engineering (ICDE)*, 2006.

[50] Introduction to emmc. https://www.slideshare.net/linaroorg/intro-to-emmc.

[51] N. L. Farnan, A. J. Lee, P. K. Chrysanthis, and T. Yu. Don't reveal my intension: Protecting user privacy using declarative preferences during distributed query processing. In *Proceedings of the 16th European Conference on Research in Computer Security (ESORICS)*, 2011.

[52] N. L. Farnan, A. J. Lee, P. K. Chrysanthis, and T. Yu. Paqo: A preference-aware query optimizer for postgresql. *Proc. VLDB Endow.*, 2013.

[53] The cnil's restricted committee imposes a financial penalty of 50 million euros against google llc. https://www.cnil.fr/en/cnils-restricted-committee-imposes-financial-penalty-50-million-euros-against-google-llc.

[54] D. Garg and F. Pfenning. A proof-carrying file system. In *IEEE Symposium on Security and Privacy (S&P)*, 2010.

[55] Data deletion on google cloud platform. https://cloud.google.com/security/deletion/.

[56] Google cloud. https://cloud.google.com/.

[57] Introducing Google Cloud Confidential Computing with Confidential VMs. https://cloud.google.com/blog/products/identity - security/introducing - google-cloud-confidential-computing-with-confidential-vms.

[58] F. Gregor, W. Ozga, S. Vaucher, R. Pires, D. L. Quoc, S. Arnautov, A. Martin, V. Schiavoni, P. Felber, and C. Fetzer. Trust management as a service: Enabling trusted execution in the face of byzantine stakeholders. In *International Conference on Dependable Systems and Networks, (DSN 2020)*, 2020.

[59] B. Gu, A. S. Yoon, D.-H. Bae, I. Jo, J. Lee, J. Yoon, J.-U. Kang, M. Kwon, C. Yoon, S. Cho, and et al. Biscuit: A framework for near-data processing of big data workloads. In *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA)*, 2016.

[60] B. Gu, A. S. Yoon, D.-H. Bae, I. Jo, J. Lee, J. Yoon, J.-U. Kang, M. Kwon, C. Yoon, S. Cho, J. Jeong, and D. Chang. Biscuit: A framework for near-data processing of big data workloads. In *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA*, 2016.

[61] M. Hähnel, W. Cui, and M. Peinado. High-resolution side channels for untrusted operating systems. In *2017 USENIX Annual Technical Conference (USENIX ATC)*, 2017.

[62] A. Y. Halevy. Answering queries using views: A survey. *The VLDB Journal*, 2001.

[63] Taishan 200 server, 5290 storage model. https://e.huawei.com/uk/products/servers/taishan-server/taishan-5290.

[64] Data-in-use protection on IBM Cloud using Intel SGX. https://www.ibm.com/cloud/blog/data-use-protection-ibm-cloud-using-intel-sgx.

[65] Intel corporation. attestation service for intel software guardextensions (intel sgx): Api documentation. https://api.trustedservices.intel.com/documents/sgx-attestation-api-spec.pdf.

[66] Intel software guard extensions (intel sgx). https://software.intel.com/en-us/sgx.

[67] Z. István, S. Ponnapalli, and V. Chidambaram. Software-defined data protection: Low overhead policy compliance at the storage layer is within reach! *Proc. VLDB Endow.*, 2021.

[68] M. Jayashree, W. Melissa, and C. Vijay. Analyzing GDPR compliance through the lens of privacy policy. In *Heterogeneous Data Management, Polystores, and Analytics for Healthcare - VLDB Workshops, Poly and DMAH*, 2019.

[69] I. Jo, D.-H. Bae, A. S. Yoon, J.-U. Kang, S. Cho, D. D. G. Lee, and J. Jeong. Yoursql: A high-performance database system leveraging in-storage computing. *Proc. VLDB Endow.*, 9(12):924–935, Aug. 2016.

[70] T. Kim, J. Park, J. Woo, S. Jeon, and J. Huh. Shieldstore: Shielded in-memory key-value storage with sgx. In *Proceedings of the Fourteenth EuroSys Conference 2019 (EuroSys)*, 2019.

[71] G. Koo, K. K. Matam, T. I, H. V. K. G. Narra, J. Li, H.-W. Tseng, S. Swanson, and M. Annavaram. Summarizer: Trading communication with computing near storage. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2017.

[72] I. Kotsogiannis, Y. Tao, X. He, M. Fanaeepour, A. Machanavajjhala, M. Hay, and G. Miklau. Privatesql: A differentially private sql query engine. *Proc. VLDB Endow.*, 2019.

[73] R. Krahn, B. Trach, A. Vahldiek-Oberwagner, T. Knauth, P. Bhatotia, and C. Fetzer. Pesos: Policy enhanced secure object store. In *Proceedings of the Thirteenth European Conference on Computer Systems (EuroSys)*, 2018.

[74] T. Kraska, M. Stonebraker, M. Brodie, S. Servan-Schreiber, and D. J. Weitzner. Datumdb: A data protection database proposal. In *Poly'19 co-located at VLDB 2019*, 2019.

[75] T. Kraska, M. Stonebraker, M. L. Brodie, S. Servan-Schreiber, and D. J. Weitzner. Schengendb: A data protection database proposal. In V. Gadepally, T. G. Mattson, M. Stonebraker, F. Wang, G. Luo, Y. Laing, and A. Dubovitskaya, editors, *Heterogeneous Data Management, Polystores, and Analytics for Healthcare - VLDB 2019 Workshops, Poly and DMAH*, 2019.

[76] D. Le Quoc, F. Gregor, J. Singh, and C. Fetzer. Sgx-pyspark: Secure distributed data analytics. In *The World Wide Web Conference (WWW)*, 2019.

[77] K. LeFevre, R. Agrawal, V. Ercegovac, R. Ramakrishnan, Y. Xu, and D. DeWitt. Limiting disclosure in hippocratic databases. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases (VLDB)*, 2004.

[78] H. Li, M. Hao, S. Novakovic, V. Gogte, S. Govindan, D. Ports, I. Zhang, R. Bianchini, H. S. Gunawi, and A. Badam. Leapio: Efficient and portable virtual nvme storage on arm socs. In *Proceedings on the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2020.

[79] X. Li, H. Hu, G. Bai, Y. Jia, Z. Liang, and P. Saxena. Droidvault: A trusted data vault for android devices. In *19th International Conference on Engineering of Complex Computer Systems (ICECCS)*, 2014.

[80] F. D. McSherry. Privacy integrated queries: An extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2009.

[81] A. Mehta, E. Elnikety, K. Harvey, D. Garg, and P. Druschel. Qapla: Policy compliance for database-backed systems. In *26th USENIX Security Symposium (USENIX Security)*, 2017.

[82] C. Mihali, A. Hangan, G. Sebestyen, and Z. István. The case for adding privacy-related offloading to smart storage. In *Proceedings of the 14th ACM International Conference on Systems and Storage (SYSTOR)*, 2021.

[83] P. Mishra, R. Poddar, J. Chen, A. Chiesa, and R. A. Popa. Oblix: An efficient oblivious search index. In *IEEE Symposium on Security and Privacy (S&P)*, 2018.

[84] A. Motro. An access authorization model for relational databases based on algebraic manipulation of view definitions. In *Proceedings of the Fifth International Conference on Data Engineering (ICDE)*, 1989.

[85] N. Santos, R. Rodrigues, K. P. Gummadi, and S. Saroiu. Towards trusted cloud computing. In *Proceedings of the 1st USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2009.

[86] A. Narayan and A. Haeberlen. Djoin: Differentially private join queries over distributed databases. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation (OSDI)*, 2012.

[87] NGD Systems. https://www.ngdsystems.com/, 2020.

[88] Eideticom. https://www.eideticom.com/. Last accessed: Jan 2020.

[89] NVMe Specifications. https://nvmexpress.org/specifications/. Last accessed: Jan 2020.

[90] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa. Oblivious multi-party machine learning on trusted processors. In *Proceedings of the 25th USENIX Conference on Security Symposium (USENIX Security)*, 2016.

[91] Secure storage. https://optee.readthedocs.io/en/latest/architecture/secure_storage.html.

[92] Open portable trusted execution environment. https://www.op-tee.org/.

[93] Oracle Exadata. https://www.oracle.com/engineered-systems/exadata/, 2020.

[94] A. Papadimitriou, R. Bhagwan, N. Chandran, R. Ramjee, A. Haeberlen, H. Singh, A. Modi, and S. Badrinarayanan. Big data analytics over encrypted datasets with seabed. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI)*, 2016.

[95] S. Pinto and N. Santos. Demystifying arm trustzone: A comprehensive survey. *ACM Computing Surveys (CSUR)*, 2019.

[96] R. Poddar, T. Boelter, and R. A. Popa. Arx: An encrypted database using semantically secure encryption. *Proc. VLDB Endow.*, 12(11):1664–1678, July 2019.

[97] R. Poddar, C. Lan, R. A. Popa, and S. Ratnasamy. Safebricks: Shielding network functions in the cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2018.

[98] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. Cryptdb: Protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles (SOSP)*, 2011.

[99] C. Priebe, K. Vaswani, and M. Costa. Enclavedb: A secure database using sgx. In *IEEE Symposium on Security and Privacy (S&P)*, 2018.

[100] D. L. Quoc, F. Gregor, S. Arnautov, R. Kunkel, P. Bhatotia, and C. Fetzer. Securetf: A secure tensorflow framework. In *Proceedings of the 21st International Middleware Conference (Middleware)*, 2020.

[101] G. D. P. Regulation. Regulation eu 2016/679 of the european parliament and of the council of 27 april 2016. *Official Journal of the European Union. Available at: http://ec. europa. eu/justice/data-protection/reform/files/regulation_oj_en. pdf (accessed 20 September 2017)*, 2016.

[102] RISC-V. Keystone Open-source Secure Hardware Enclave. https://keystone-enclave.org/.

[103] S. Rizvi, A. Mendelzon, S. Sudarshan, and P. Roy. Extending query rewriting techniques for fine-grained access control. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, 2004.

[104] N. Santos, K. P. Gummadi, and R. Rodrigues. Towards trusted cloud computing. In *Proceedings of the 2009 Conference on Hot Topics in Cloud Computing (HotCloud)*, 2009.

[105] N. Santos, H. Raj, S. Saroiu, and A. Wolman. Using arm trustzone to build a trusted language runtime for mobile applications. In *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems (ASPLOS)*, 2014.

[106] N. Santos, R. Rodrigues, and B. Ford. Enhancing the os against security threats in system administration. In *Proceedings of the 13th International Middleware Conference (Middleware)*, 2012.

[107] N. Santos, R. Rodrigues, K. P. Gummadi, and S. Saroiu. Policy-sealed data: A new abstraction for building trusted cloud services. In *21st USENIX Security Symposium (USENIX Security)*, 2012.

[108] 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.

[109] S. Shastri, M. Wasserman, and V. Chidambaram. Gdpr anti-patterns. *Commun. ACM*, 2021.

[110] R. Shay, U. Blumenthal, V. Gadepally, A. Hamlin, J. D. Mitchell, and R. K. Cunningham. Don't even ask: Database access control through query control. *SIGMOD Rec.*, 2019.

[111] A. Singh and K. Chatterjee. Cloud security issues and challenges: A survey. *Journal of Network and Computer Applications*, 79:88–115, 2017.

[112] Samsung. https://www.nimbix.net/wp-content/uploads/2019/07/SmartSSD_Product_Brief_Digital.pdf. Last accessed: Jan 2020.

[113] SQLite. https://www.sqlite.org/.

[114] SQLCipher. https://www.zetetic.net/sqlcipher/.

[115] S. Supreeth, W. Melissa, and C. Vijay. GDPR anti-patterns: How design and operation of modern cloud-scale systems conflict with GDPR. *CoRR*, 2019.

[116] S. Supreeth, W. Melissa, and C. Vijay. The seven sins of personal-data processing systems under GDPR. In *11th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2019.

[117] S. Supreeth, B. Vinay, W. Melissa, K. Arun, and C. Vijay. Understanding and benchmarking the impact of GDPR on database systems. *Proceedings of the VLDB Endowment*, 2020.

[118] Introduction to Trusted Execution Environments. https://globalplatform.org/wp-content/uploads/2018/05/Introduction-to-Trusted-Execution-Environment-15May2018.pdf.

[119] J. Thalheim, H. Unnibhavi, C. Priebe, P. Bhatotia, and P. Pietzuch. Rkt-io: A direct i/o stack for shielded execution. In *Proceedings of the Sixteenth European Conference on Computer Systems (EuroSys)*, 2021.

[120] TPC-H. http://www.tpc.org/tpch/.

[121] B. Trach, R. Faqeh, O. Oleksenko, W. Ozga, P. Bhatotia, and C. Fetzer. T-lease: A trusted lease primitive for distributed systems. In *Proceedings of the 11th ACM Symposium on Cloud Computing (SoCC)*, 2020.

[122] B. Trach, A. Krohmer, F. Gregor, S. Arnautov, P. Bhatotia, and C. Fetzer. Shieldbox: Secure middleboxes using shielded execution. In *Proceedings of the Symposium on SDN Research (SOSR)*, 2018.

[123] B. Trach, O. Oleksenko, F. Gregor, P. Bhatotia, and C. Fetzer. Clemmys: Towards secure remote execution in faas. In *Proceedings of the 12th ACM International Conference on Systems and Storage (SYSTOR)*, 2019.

[124] S. Tu, M. F. Kaashoek, S. Madden, and N. Zeldovich. Processing analytical queries over encrypted data. *Proceedings of the VLDB Endowment*, 2013.

[125] A. Vahldiek-Oberwagner, E. Elnikety, A. Mehta, D. Garg, P. Druschel, R. Rodrigues, J. Gehrke, and A. Post. Guardat: Enforcing data policies at the storage layer. In *Proceedings of the Tenth European Conference on Computer Systems (EuroSys)*, 2015.

[126] N. Volgushev, M. Schwarzkopf, B. Getchell, M. Varia, A. Lapets, and A. Bestavros. Conclave: Secure multi-party computation on big data. In *Proceedings of the Fourteenth EuroSys Conference (EuroSys)*, 2019.

[127] S. Volos, K. Vaswani, and R. Bruno. Graviton: Trusted execution environments on gpus. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2018.

[128] Y. Xu, W. Cui, and M. Peinado. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *IEEE Symposium on Security and Privacy (S&P)*, 2015.

[129] M. H. Yun and L. Zhong. Ginseng: Keeping secrets in registers when you distrust the operating system. In *26th Annual Network and Distributed System Security Symposium (NDSS)*, 2019.

[130] W. Zheng, A. Dave, J. G. Beekman, R. A. Popa, J. E. Gonzalez, and I. Stoica. Opaque: An oblivious and encrypted distributed analytics platform. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation (NSDI)*, 2017.

[131] J. Zhu, R. Hou, X. Wang, W. Wang, J. Cao, B. Zhao, Z. Wang, Y. Zhang, J. Ying, L. Zhang, et al. Enabling rack-scale confidential computing using heterogeneous trusted execution environment. In *IEEE Symposium on Security and Privacy (S&P)*, 2020.