Privacy-preserving third-party computations on secure personal data management systems

Robin Carpentier

université
PARIS-SACLAY

Privacy-preserving third-party computations on secure personal data management systems

Calculs tiers respectueux de la vie privée sur des systèmes de gestion de données personnels et sécurisés

Thèse de doctorat de l'université Paris-Saclay

École doctorale n°580 : Sciences et Technologies de l'Information et de la Communication (STIC)
Spécialité de doctorat: Informatique
Graduate School : Informatique et Sciences du Numérique,
Référent : Université de Versailles-Saint-Quentin-en-Yvelines

Thèse préparée dans l'unité de recherche DAVID (Université Paris-Saclay, UVSQ, Inria), sous la direction de **Nicolas ANCIAUX**, Directeur de recherche, et le co-encadrement de **Iulian SANDU POPA**, Maître de conférences.

Thèse soutenue à Versailles, le 14 Décembre 2022, par

Robin CARPENTIER

Composition du jury

Membres du jury avec voix délibérative

Prénom Nom Titre, Affiliation	Président ou Présidente
Reza Akbarinia Chargé de recherche, HDR, Inria, Université de Montpellier	Rapporteur & Examinateur
Sébastien Gambs Professeur, Université du Québec à Montréal	Rapporteur & Examinateur
Luc Bouganim Directeur de recherche, Inria, Université Paris-Saclay	Examinateur
Mathieu Cunche Maître de conférences, HDR, Inria, INSA Lyon	Examinateur
Benjamin Nguyen Professeur, INSA Centre Val de Loire	Examinateur
Patricia Serrano Alvarado Maîtresse de conférences, HDR, Nantes Université	Examinatrice

THESE DE DOCTORAT
NNT : 2022UPASG079

université
PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies
de l'information et de
la communication (STIC)

Contents

List of Figures

1 - Introduction

With technology being an essential part of our daily lives, the amount of data generated worldwide is enormous and growing steadily, namely 90 million terabytes created per day in 2018 and estimated to reach 480 million terabytes by 2025 [1]. This massive generation of data is partially fueled by the growing number of individuals having access to the internet, with predictions reaching 66 percent of the global population by 2023, and also by the multiplication of connected devices with an estimated 29.3 billion networked devices on the same year [2].

In this data pool, any information related to an identified or identifiable natural person is labeled *personal data* [3]. Resulting from our interactions with our devices (e.g., smartphones, computers, voice assistants) and online services (e.g. social media, e-shopping, video games), fulfillment of administrative procedures, purchases and other financial transactions, medical acts, use of transports, communications, etc., such data about ourselves are created on a daily basis by many different entities.

On the one hand, gathering data about individuals enables a better understanding of our society and fuels progress. Thanks to ever more sophisticated algorithms, these data can provide a high degree of insights to improve many aspects of our daily lives, for example how we move [4, 5], how we eat [6, 7], how we grasp relationships [8], how local or national governments oversee urban planning [9] or anticipate energy demand and promote its conservation [10], and how companies evaluate market and improve their product [11].

On the other hand, this much knowledge about individuals can also become a concern if not properly managed. Over time, most of these personal data have been centralized by companies owning the products or services that create them. While it is convenient for them to easily access and benefit from this data pool, we consider this classical model to have major flaws. First, it attracts the scrutiny of attackers, worsening the impact of a successful attack which can compromise the data of millions, if not billions of individuals. These data breaches are not a phantom threat and are in fact common events in the digital world [12]. Aside from the loss of trust incurred by the companies, they have disastrous consequences on individuals: they lead to identity thefts, fraudulent payments, phishing attempts etc.

Furthermore, it has been revealed that multinational corporations which are storing vast amount of personal data are involved in global surveillance schemes [13] implemented by some states as part of their defense and intelligence policies. The unprecedented revelations by Edward Snowden in 2013 [14] about how corporations of the web and communication industries collaborated with law enforcement agencies by handing over the personal data of millions of citizens, but also about how countries signed secret treaties to share this kind of intelligence,

have raised awareness and distrust about the centralization model.

Finally, in the hands of these companies, personal data is subject to concealed exploitation which is not always understood or consented by users and can impact negatively their experience with technology or their lives. Indeed, it has been reported that these data are shared and collected by companies in the goal of establishing extremely accurate profiles about individuals [15]. These profiles store values and scores about many different aspects of the life of their target, including "education, employment, political views, ethnicity, religion, health interests, media usage, purchases, income, economic stability, personality [...] website visits, video views, app usage, movements, and web searches, including sensitive 'interests' [...]" [15]. Afterwards, these profiles are used for a wide range of questionable motives, for example to predict individual creditworthiness or to maximize customers profitability by personalized telemarketing scripts, online content or ads and more generally "to persuade consumers to act in certain ways" [15].

Considering these flaws, protecting personal data has become a concern to many people [16], a necessity for some. Fortunately, initiatives have been launched to help individuals in this regard. In 2010, a governmental initiative called The Blue Button enable citizens of the United States of America to download their health data in a portable and readable format. In 2018, the European Council introduced the General Data Protection Regulation (GDPR) [3], which grants all individuals in the European Union substantial rights regarding the handling of their personal data. GDPR regulates how companies around the world collect and process data about European citizens, whether these firms are based in Europe or elsewhere. In a nutshell, citizens are given the right to access, erase or restrict the processing of their personal data, as well as the right to data portability. At a lower scale, the California Consumer Privacy Act (CCPA) [17] signed into law the same year and the California Privacy Rights Act (CPRA) [18] effective in 2023 grant a subset of these rights to citizens of the state of California in the United States of America. Outside of the legal sphere, it is also worth mentioning that certain products and organizations have made the privacy of their users a central component. For example, Signal [19] is an instant messaging application whose primary claims are to employ strong cryptography and near-zero data collection of their users; Proton [20] is a suite of web services (mail, calendar, cloud storage and VPN) where privacy is the watchword of their development; and Framasoft [21] is a french not-for-profit organization freely providing open-source alternatives to some popular services with an incentive on privacy.

From the newly granted rights cited above, the right to data portability [3, Article 20] is of particular interest for this thesis. Data portability follows the idea of The Blue Button initiative by allowing individuals to retrieve a copy of their personal data held by companies and administrations. It goes beyond this previous initiative by expanding this ability, not only to health data, but to all companies handling personal data. This data retrieval must be carried out "in a structured,

commonly used and machine-readable format," which is essential for users to be of practical use. However, without the proper tools to manage it themselves, data portability might only be used to be handed over to another entity, where the problems of centralization emerge once again.

On that account, appropriate solutions have spawned to fulfill this role. *Personal Data Management Systems* (PDMSs) arise from industrial and scientific proposals [22, 23] aiming to empower individuals with their own data. They are shaped as hardware or software user-centered assistants intending to support them in their digital life, including automatic data gathering, informed data sharing and beneficial data computing while providing data security and ease of use. PDMSs implement an interesting new paradigm for data-driven computations: instead of sharing an entire batch of data to be computed by a third party, a specialized data processing code written by third parties, can be imported on the PDMS to be executed. Only the result of the computation (but not the raw personal data used as input) is shared with the third party. As such, a user will still be able to receive third-party benefits without ceding their data. The examples below illustrate this new paradigm.

> ***'Green bonus' example.*** Companies want to reward their employees having an ecological conduct, and thus monthly compute a green bonus (financial incentive) based on the number of commutes by bike. To this end, an employee collects their GPS traces within their PDMS from a reliable service (e.g., Google Maps). The traces are then processed locally and the result is delivered to the employer with a compliance proof.

> ***'Energy' example.*** An energy supplier wishes to offer services, precisely calibrated according to the energy consumption of its future customers. In order to establish a tailor-made offer, the provider needs to evaluate different statistical computations on the customer's consumption. Thanks to their PDMS offering confidentiality guarantees and a smart meter, customers agree to disclose these statistics but not their detailed consumption, and thus install the function sent by the supplier. The attestations provided by the PDMS even allow the supplier to commit to a quote since they obtain guarantees on the computation made.

Several similar scenarios can be envisioned for service offers based on statistical analysis of historical personal *factual* data related to a user, e.g., health services (medical and prescription history), car insurance (GPS traces), financial services (bank records).

In order to be viable and support these scenarios, we argue that PDMS solutions must meet two fundamental requirements. First, they must cover the entire data life cycle including their collection, storage, management, sharing and processing. This completeness is essential to replace centralized solutions and ensure that users will not need to rely on another third-party provider. PDMSs must be designed

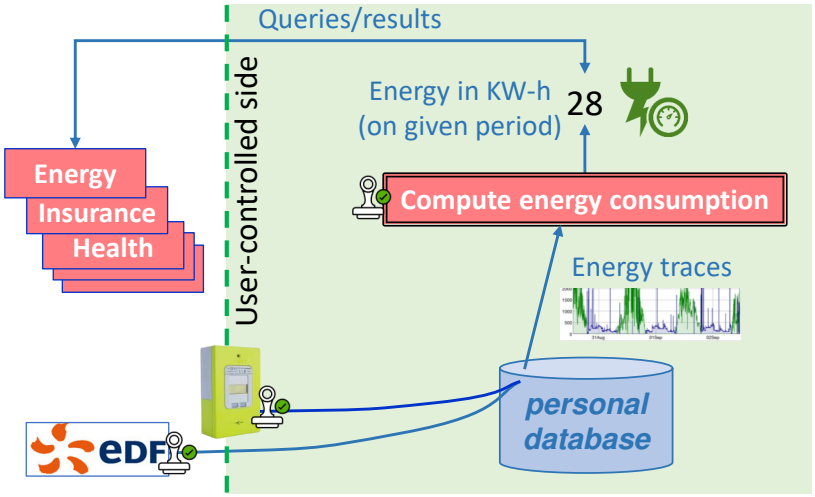


Figure 1.1: PDMS computation scenario ('Energy')

to adapt easily to the many different types of data and computations they will encounter, and should therefore implement *extensiveness*. Second, the *security* of PDMSs is paramount. They must incorporate the necessary mechanisms to protect potentially the entire digital life of their owner while requiring very little knowledge about computer security, the owner being a layman. In our Green Bonus example, employees must be assured that their detailed GPS data is not disclosed to third parties (including their employer) outside of the sphere of control of their PDMS. PDMSs must also be able to prove the veracity of the result to the third party, e.g., attesting to the employer the genuineness of the computation. Extensiveness and security are conflicting properties and designing such solution is challenging.

The advent of *Trusted Execution Environments* (TEEs) have given new hopes of achieving this goal. TEEs are hardware and/or software solutions allowing to run a program on a machine in an isolated way. This isolation is characterized by a protection against tamper of said program and against observation of the processed data. Moreover, they also include an attestation mechanism, certifying the identity of executed program. Recently, TEEs have been included inside many different models of processors embedded in smartphones, computers and servers, opening new possibilities of ensuring security on a potentially insecure device.

Objective. Our goal is to solve this conflict between extensiveness and security in the PDMS context for computations dealing with large volumes of personal data (typically, aggregation functions). Extensiveness is made possible by allowing the computation code to be written by a third-party entity querying the PDMS. This code will be evaluated in the PDMS environment, but it cannot be considered fully trusted from the point of view of the *PDMS user* (owner of the data and of the PDMS). In this setting, this thesis focuses on controlling the potential information leakage that can occur through the computation results. Indeed, as the computation code –written by a third party– will get access to personal data

and produce a result based on said data, –result which will be sent to a third party–, leakage of information can occur especially during successive executions of such computations.

Contributions. This thesis exposes the following contributions:

- An Extensive and Secure Personal Data Management System (ES-PDMS) architecture is coupled with a widespread Trusted Execution Environment (TEE) named Intel SGX. This architecture design provides strong protection of the user's data while ensuring extensiveness on the types of computations supported, by leveraging third-party computation codes. We confront this architecture to a powerful attacker who creates these computation codes and is allowed to receive aggregate results of the executions of these codes on user's data. Then, we propose three design rules named *security building blocks* to be enforced during computations to fight against the potential leakage of user's data through the results. With these rules, we provide an upper-bound on the amount of data that can be leaked.

- We propose *execution strategies* and their associated algorithms to enhance the performances of our solutions while maintaining leakage to a minimum.

- We present an implementation prototype of a Personal Data Management System using Intel SGX. This prototype is used to make a detailed performance evaluation of our proposals.

This thesis is divided into eight chapters. The current first chapter provides the context and introduction to our work.

Chapter 2 presents the related works in order to position this thesis regarding the current state of the art on different fields. After an overview of the different types of Personal Data Management Systems and Personal Clouds, we define and expose the properties of existing Trusted Execution Environments, followed by solutions bringing together Database Management Systems and Secure Hardware. Then, we study how is the security of User Defined Functions traditionally handled. Finally, we introduce and position our work regarding alternative solutions to protect the privacy of individuals, including homomorphic encryption, anonymization and information flow analysis.

Chapter 3 describes the considered PDMS architecture with its security properties and associated computing and threats models. Then, it formulates the specific problem broached by this thesis and define leakage metrics used in the following chapters.

Chapter 4 is dedicated to our *security building blocks* and analyses their impact on data leakage. It is based on a preliminary work as a poster [24] published at the 6th IEEE European Symposium on Security and Privacy (EuroS&P 2021), then described in detail [25] at the 11th International Conference on Data Science, Technology and Applications (DATA 2022).

The inner functioning of our execution strategies mentioned above is shown in Chapter 5. They were introduced in [26] at the 34th International Conference on Scientific and Statistical Database Management (SSDBM 2022) and also presented at the 38th French Conference on Data Management - Principles, Technologies and Applications (BDA 2022) [27].

Chapter 6 is devoted to the performance analysis of the proposals of the previous two chapters. It has also been presented in [26] and [27].

Our implementation prototype and its demonstration features are covered by Chapter 7. It was first exposed [28] at the 25th International Conference on Extending Database Technology (EDBT 2022), then presented at the 2022 CNIL's International Privacy Research Day [29] and at the 38th French Conference on Data Management - Principles, Technologies and Applications (BDA 2022) [30].

In the end, Chapter 8 brings a conclusion to this thesis' work by summarizing the contributions and suggesting compelling future works to expand it.

2 - Related Works

Contents

This thesis is at the crossroads of data management, trusted execution environments (TEEs) and data leakage. We study a data leakage problem caused by the reliance in an untrusted computation code having access to sensitive data and we rely on TEEs to build our solutions. We focus on a context around Personal Data Management Systems (PDMSs), without excluding applications in other contexts, e.g., a traditional Database Management Systems (DBMSs) running an untrusted User-Defined function (UDF). In this chapter, we discuss existing work related to PDMSs, TEEs, the security of DBMSs and associated UDFs. We also position our work regarding existing methods of privacy protections, namely homomorphic encryption, local differential privacy, information flow analysis and network traffic analysis. Finally, we distinguish our considered problem from inference attacks.

2.1 . Personal Data Management Systems

This thesis is focused on the context of handling individuals' personal data. Over the last ten years, the treatment of such data has been the subject of much controversy and, as such, the topic of several research and industrial works. While there are many ways to store personal data, their over-collection, mass centralization, and obscure management and sharing by companies have led to the creation of user-centered data storage and management solutions. They follow the directions suggested by previous works [31, 32, 33, 34] that citizens should be empowered by tools to responsibly manage their personal data. These solutions have been given different names: Personal Data Management Systems (PDMS), Personal Clouds, Personal Data Stores or Personal Information Management Systems. Overall, they are hardware and/or software platforms enabling users to store and exploit all types of documents and data ranging from official papers like bank statements, energy bills or pay slips, to personal photos, videos, medical prescriptions or purchase receipts together with IoT-generated data such as heart rate measurements or GPS locations history. This section provides an overview of the different types of Personal Clouds grouped by the main features and privacy protection they offer.

2.1.1 . Traditional Online Personal Clouds

This first type of Personal Clouds is also the most widespread in the number of solutions. Indeed, storing data on behalf of individuals is a common business with solutions such as Google Drive [35], DropBox [36], OneDrive [37] and many more [38, 39, 40, 41, 42, 43] which enable the user to upload their data on the provider's infrastructure and access it easily afterwards via its personal devices (computer, smartphone etc ...). The main features offered by them are:

- **Accessibility.** The data of the user is stored online and as such can be accessed easily through any device connected to the Internet. Many providers claim to have a high degree (>99%) of service availability.

- **Backup and file versioning.** In case of data loss from erroneous erasure or entire device failure, the user can restore their data from the copy stored on the Cloud. Some services even provide file versioning, storing every different states caused by every modifications of files.
- **Simple data sharing.** One of the most basic feature of a Personal Cloud is to be able to share the data stored on it. This is usually achieved by the creation of a unique URL per file or folder to be shared, that a recipient can open with a web browser.
- **Collaborative editing.** Some solutions [35, 37, 43] enable users to edit some types of files collaboratively at the same time via a user interface.

Security and Trust. When dealing with security, and by extension with privacy, it is important to define who and what does the user have to trust in order to have confidence using the given technology. The provider of an individual's personal cloud potentially holds their entire digital life on their own infrastructure, and must thus gain a strong users' trust. As such, the security measures taken by the provider must be strong enough to protect users' data from unauthorized accesses and data leaks. In this way, providers of traditional online personal clouds commonly claim to follow security standards for handling data, ranging from user authentication, data transmission during upload or display on the user's device, to data storage. This is achieved through the usage of reliable cryptography primitives and protocols internationally ratified for their security. Because the provider is in charge of every aspect of this, an implicit assumption is that the provider's employees, particularly the administrators, are assumed to be fully-honest.

While this family of solutions are the most well known to the general public, they suffer from major limitations:

1. **Privacy.** The main issue is their position regarding privacy. While the data is secured using cryptography, the provider possesses the encryption key making the data subject to *secondary usages* i.e. usages not strictly required to provide the service to users but which may be of financial interest to the provider, such as data monetization. Some providers are publicly known to actively participate in the business of selling data about individuals as one of their main business model.
2. **Data management features.** The limited features proposed by these solutions constitute the second main issue. Indeed, their primary goal is to store the data of an individual and provide backups in case of accidents. However, in our opinion storage and backups are only a fraction of the features constituting the life cycle of personal data. For example, the lack of support for easily collecting data from the numerous personal data sources of an individual is problematic. From administrations delivering tax assessments and other official documents, pay slips from employers, consumption

data from energy providers, statements from financial institutions, photos and videos from social media to the huge diversity of data produced by IoT, data collection is a laborious task if done manually, given the number of entities that produce data about ourselves. Nonetheless, it is an essential starting point to be able to benefit from computations on these data, and as such should be assisted. Another feature missing is being able to benefit from rich and diverse computations in the cloud environment.

2.1.2 . No-Knowledge Online Personal Clouds

Another family of personal cloud solutions try to patch some of the weaknesses previously described by implementing architectural variations to partly alleviate the trust put on the cloud provider, like Mega [44], ProtonDrive [45], SpiderOak [46] and others [47, 48, 49, 50, 51, 52]. They are commonly characterised as 'Zero-knowledge' or 'No-knowledge'. The main particular feature highlighted by these solutions is Secure storage.

Secure storage. While traditional online personal clouds generally employs encryption of the data *at-rest*, the providers are kept in charge of managing the encryption keys of their users, meaning that they technically have the ability to read the underlying data. No-knowledge solutions move away from this possibility by making the user responsible for this task. In practice, the encryption key is derived from the user's password when they signs in. This implies that in case of password loss, the provider cannot help the user and the data are lost. The provider might still know the size and number of files stored by each user, but their content is hidden.

Security and Trust. This addition of security protects the confidentiality of data against two major attacks. An attacker gaining access to the provider's infrastructure and storage will only be able to retrieve encrypted data. Likewise, a malicious or honest-but-curious cloud provider won't be able to make secretly (i.e. non-advertised) secondary usages.

When compared with traditional online personal clouds, no-knowledge personal clouds have a higher level of security for the user because their data is not accessible by the provider in plaintext. However, these solutions share the same limitations as the previous family in terms of data management features: the lack of assisted data collection or rich data computation ecosystem. Even worse, these are partially inherent limitations of the solutions. Indeed, the provider's architecture storing data in their encrypted form, it is difficult to include processing or advanced services on it (see Section 2.5.1 for related works on the topic of computing encrypted data). If computations cannot be delegated to the provider's infrastructure, data has to be sent to the user's device for decryption before any processing can be done. In the end, the lack of trust in the provider which was the starting point of these solutions has now shifted the security and privacy issues to the client device, which is not administrated by security experts and thus more subject to viruses and other spyware programs.

2.1.3 . Privacy-Aware Personal Data Management Systems

Privacy-Aware Personal Data Management Systems go one step further with additional features aiming at covering more ground in the personal data life cycle while advertising the protection of the privacy of their user. Industrial products such as Cozy Cloud [22], Digi.me [53], Solid [32, 54] and others [55, 56, 57, 58, 59] offer individuals a digital space for their personal data on an online Cloud whose architecture is controlled and managed by the provider but with a emphasis on privacy and user consent. Indeed, a common ground of the privacy policy of these solutions is to assert that the data hosted by the provider will suffer no secondary usages that has not been explicitely consented by the user, particularly on the points of data sharing and monetization. Notable features that are implemented by some of the vendors are:

- **Data collection.** The users have access to modules which enable them to automatically import documents and other data from online services directly into their personal space. These modules are called Data Collectors and are the bridges between the Personal Cloud and data sources such as financial institutions, energy providers, internet and mobile carriers, social networks, mobility companions (e.g. geolocation history), and many more. Collectors are generally based on web scrapping, posing as the user by authenticating on the data source's website with their credentials and collecting relevant information, potentially converting it into a usable format. Recently, the usage of collectors has been improved by the introduction of regulations like the GDPR in the European Union or the CCPA in California, United States –particularly the right to data portability–, and smart disclosure initiatives (e.g. BlueButton, GreenButton and MyStudentData in the United States).

- **Cross-data computations.** Given that the data have been gathered inside its Personal Cloud, the user can now benefit from computations crossing the information from different data sources together, enabling new exploitation and analysis that was previously inaccessible. For example, CozyCloud allows applications to be run on the user's personal cloud. Cozy facilitates the recognition and classification of documents by publishing a list of available doctypes to application developers. User documents are associated with a doctype family (e.g, bank, photo, bills etc.) such that applications can easily be granted access to and cross-exploit documents.

- **Advanced data sharing.** Another important aspect taking advantage of this pool of data is the ability to share it easily with third parties in order to benefit from various advantages (cashbacks, loans etc.). Beyond the basic document sharing of traditional solutions, Digi.me offers for example an ecosystem for individuals and businesses to efficiently share and receive data, in respect of the user consent which has to be explicitly granted.

Security and Trust. Individuals being more and more wary about how their personal data is handled, a key adoption factor is to stand out from the traditional solutions and answer the users' call for privacy. To gain the trust of privacy-concerned users, the providers usually claim to adhere to a variety of privacy morals and security specifications. A common claim is to commit to the users' sole ownership and control of their data, by promising to never observe, exploit or share data for *secondary usages*. Beyond the argument of following security standards like traditional solutions, they usually rely on two arguments :

- **Virtuous legal and economic framework.** Users and providers agree to a contract (usually named 'Terms of Service' and 'Privacy Policy') which, among other things, precisely defines how is data collected and used by the provider. This is a more formal presentation of their intentions which depicts the business model they adopted, particularly regarding the non sharing and selling of personal data.

- **Transparency.** Most providers claim to propose transparency and/or accessibility of the code base constituting their solution. Some of them [22, 58] have open sourced their code which allows anyone to independently check the strength of the security claims. Others [53, 57] have their code regularly audited by independent security experts whose reports may be released as a transparency decision.

To wrap up, this family of solutions comprises features to handle the collection of personal data, their storage, secured access, and their processing to gain knowledge by bringing data sources together and share it with third parties. Earning users' trust to use the product is paramount, and providers make strong privacy promises and claim a business model free of data selling and respectful of the user's privacy. Theses claims are also the main weakness of these solutions: it is unclear how binding they are to the technical means meant to enforce them. They imply strong security hypotheses, for instance the provider's employees, particularly the administrators, are assumed to be fully-honest. All portions of code, from the underlying storage mechanism to the entire set of applications and services running on top of it, have to be trusted. Because the data is centralized on the provider's infrastructure, a successful attack on it or an error, negligence or corruption of one the employee could result in the leakage of a vast amount of data pertaining to multiple individuals.

2.1.4 . Home PDMS Software

A third family of Personal Cloud is Home PDMS Software. The main difference with their online counterpart is the location of the data: it is not stored on an online infrastructure anymore but instead on a device that is owned by the user, ideally close to the IoT devices which produce the data (e.g., their personal computer). This is an approach based on the paradigm of local/edge computing which leverages

the edges of the network instead of centralizing the data of many individuals in the same place. This family includes for instance OpenPDS [60], DataBox [61] and Personium [62]. Some online solutions like Cozy [22] and NextCloud [58] propose self-hosted instances as well. Overall, they aim for offering advanced features previously described such as data collection, cross-data computations and advanced data sharing.

- **Cross-computations and data dissemination.** An important objective is to limit the amount of data that is shared with a third party. Instead of granting them access to large sets of data for computation, the computation can be done on the user's device containing the data and disclose only the results. To this end, OpenPDS [60] proposes for example a framework called Safe Answer, which computes the data of the owner to answer precise questions. Just as their online counterpart, they enable cross-computations of data coming from different sources.

Security and Trust. This type of PDMS being hosted on a user device, it assumes that both the device and the PDMS software are trusted. On the one hand, the entire software stack has to be trusted, including the data system implementing the aforementioned functionalities such as the software codes carrying out the query framework. Software isolation, for example with Docker containers in Databox [61], might also be included. On the other hand, the user's regular devices (e.g., their computer), in addition to all the other purposes for which it is used, will also bear the task of storing, securing and computing potentially the entire user's digital life.

Instead of relying on a third party to host their personal data, home PDMS softwares entrust the user device with this task and to carry out the rich functionalities like data sharing or computations. This family of solutions is indeed focused on querying and sharing capabilities with respect to the user's privacy. The entire software and the entire user device must be trusted, and while software isolation can be used, no formal security guarantees are offered. Because the user device serves many other purposes than hosting the PDMS, it is subject to viruses and malwares and as such might not be the ideal place to store potentially the entire user's digital life.

2.1.5 . Home PDMS Devices

To isolate the management of personal data from the other activities an individual might perform on their device, another approach is to dedicate another device to host the data at home. Amber X [63] and the now discontinued CloudLocker [64], Helixee [65] and Meet Lima [66] are examples of self-host devices that can hold hundreds of gigabytes or more if needed, while being synchronized with all the devices of the owner. In terms of features, they only mirror some the capabilities of traditional personal clouds at the user's home: accessibility via the user's home router, backup, file versioning and simple data sharing. Advanced

data management features like data collection and cross-data computations are, to the best of our knowledge, not supported, with the exception of the PRO version of Amber X which includes the possibility of installing Docker containers but is reserved to advanced users.

Security and Trust. Like home cloud software, the main privacy benefits of using home cloud devices is the non-reliance on a centralized cloud architecture. However, the same assumptions have to be made about the software and the hardware: they have to be trusted. A strengthening factor of this trust is that the device's capabilities are circumscribed to managing personal data, as opposed to general-purpose computers or smartphones a user can possess. As such, the attack surface of home cloud devices is limited.

Home cloud devices go one step further in terms of trust than their software counterpart, by devoting an entire device to the purpose of hosting the personal cloud and offering the basic set of features found in traditional personal clouds. While the attack surface is as such limited, these solutions do not provide strong security guarantees while being subject to connections from insecure end-user devices from the local network and the internet. In terms of features, the advanced data management tools described previously are missing.

2.1.6 . Portable PDMS Devices with Tamper-Resistant Hardware

Other proposals go one step further than home PDMS devices and embed a tamper-resistant hardware into a portable device. We make the distinction between, on the one hand tamper-resistant hardware such as smart-cards, secure micro-controllers and trusted platform modules (TPM), and on the other hand trusted execution environments whose applications on data management will be discussed in Section 2.3. By introducing tamper-resistant hardware, some research proposals including Personal Data Server (PDS) [67] and Trusted Cells [68] intend to strengthen the security of PDMS Devices. This hardware is a secure ship similar to that of a smart-card and hosts a minimal Trusted Computing Base (TCB) containing a Database Management System (DBMS). Applications of these solutions are usually highly specific, for example health data [69, 70, 71] or text-based documents [72] (see tutorials [73, 74]). They incorporate the following features:

- **Secure cross-computations.** SQL database queries are supported through the integration of a query evaluation and an access control engine on the PDS running inside the secure chip [72, 75].

- **Data dissemination.** The owner of the portable device is offered features to securely share information contained in their device [69, 76]. For example [76] proposes a sharing paradigm to easily visualize and specify document sharing policies.

- **Secure distributed computations.** Crossing the data of multiple individuals in a network of PDSs is an interesting feature that has been explored

[77]. While each user keeps control over their data, they can express the need to participate to statistical queries over their data. [78] and [79] use an hybrid infrastructure to compute SQL aggregates or special aggregation in a mobile participatory sensing context. They involve a powerful and honest-but-curious cloud infrastructure to offer a trade-off between privacy and performances.

Security and Trust. The addition of a secure chip to the PDMS device brings a strong security factor to these solutions compared with the previous family. Because the DBMS engine is embedded in this chip, the data are cryptographically protected. Hardware and software attacks are as such difficult to achieve. This also imply that the DBMS engine is proven secure.

A strong security level can be reached with this family of solutions. Works about PlugDB [80] contributes to many aspect of data management, from a single individual's data with SQL queries [75] or full-text searches [72, 81] to the handling of the data of many individuals with SQL queries [77], anonymisation [82], participatory sensing [79] and data exchanges through opportunistic networks [83]. However, the addition of tamper-resistant chip diminish greatly the computation capabilities of these proposals [84, 85, 86]. Indeed, the system is hardly extensible: adding new data and query models leads to a redesign of the data storage, indexing and query processing to cope with the hardware's constraints. Also, advanced data processing are not supported as not being compatible with a minimal TCB.

2.1.7 . Conclusion

Due to growing concerns about how personal data are handled by corporations, user-centered personal data storage solutions are flourishing with the objective to give users the control of their data back, particularly boosted by new privacy regulations such as the right to data portability in the GDPR [3]. Personal data management systems are a promising tool, already having many practical applications such as education [87], health [69, 88], the Internet-of-Things [72, 81, 89] and Artificial Intelligence [90]. However, we argue that the most used traditional online personal cloud solutions are not viable answers to both of these privacy concerns: either they do not meet the privacy basics, leaving the data subject to secondary usages by the cloud provider ; or they are limited to a subset of the life cycle of data, not supporting diverse computations meant to benefit from these newly gathered data. While privacy-aware personal data management system alternatives are developed and comprise advanced data processing features, they currently rely on too strong trust assumptions of their entire software stack and privacy claims of the providers.

On another hand, at-home software solutions rely on multi-purposes end-user devices to handle these features, which makes the personal data vulnerable to all the threats and vulnerabilities of such an insecure device. Dedicated devices lower the attack surface but still rely on trusting their entire software and hardware

stacks. Finally, tamper-resistant hardware devices rely on specific secure hardware (secure microcontroller –or TPM– running a lighweight DBMS engine connected to a mass-storage flash module holding the personal database) as well as a minimal Trusted Computing Base (TCB) for the PDMS, leading to increased security guarantees, but with reduced performance (due to the drastic limitations of the hardware considered) and no possibility to extend the security sphere to untrusted external processing code (which by definition, cannot be included in the TCB).

Designing a solution answering both the privacy and extensibility requirements is an open problem, addressed by this thesis. Our approach also relies on an architecture including secure hardware [23], but allows the TCB to be coupled with advanced non-secure processing modules to provide support for rich data computations and control over potential data leakage.

2.2 . Trusted Execution Environments

Bringing security to data storage and computation through secure hardware is a common approach. SIM cards, credit cards, passports and public transport passes are everyday examples of secure hardware used to securely store information and provide some computing power associated. In these examples, the secure hardware was tailored for their specific use-cases which only require a small computing power. In order to manage personal data, the computing platform must be able to adapt to many different data and computation types.

With this in mind, a promising class of secure hardware called Trusted Execution Environments (TEEs) is of particular interest. TEEs bring security to computations through the creation of a tamper-resistant processing environment [91]. The last decade has seen the rise of including TEEs inside general-public devices such as smartphones and personal computers, but also inside high-end machines like servers. Intel Software Guard Extensions (Intel SGX) [92, 93] and Arm TrustZone [94, 95] are examples of TEEs embedded inside general-public Intel or Arm CPUs. Their ubiquity and proper computing power make them prime candidates for personal data management systems.

2.2.1 . Security Properties

The following security properties are ensured by TEEs and illustrated in Figure 2.1:

- **Isolation.** The code running inside the TEE is isolated from the rest of the system, including the operating system and the hypervisor. These trusted environments are commonly called *enclaves* or *secure world* and other programs of the system cannot tamper with the code executing inside. In particular, the data processed by the isolated code is protected from eavesdropping and malicious modification, both from physical and software attacks. This is achieved by encrypting and keeping track of the integrity of the RAM used

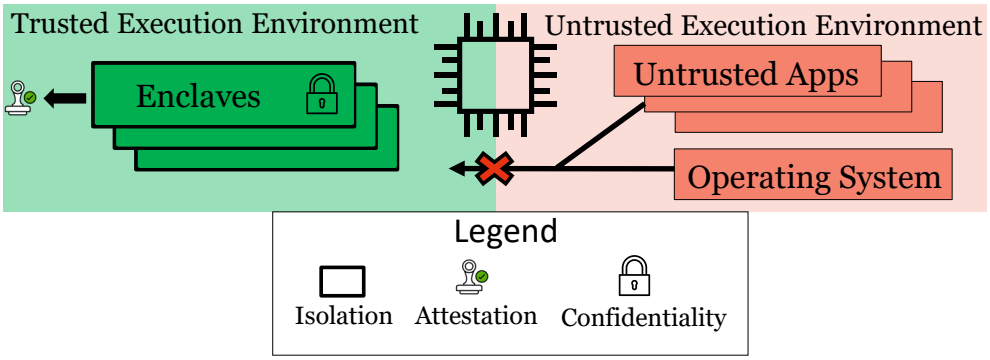


Figure 2.1: Trusted Execution Environments.

by the code [96].

- **Attestation.** While being executed inside the TEE, the isolated code is able to produce a proof of its identity certifying that it is running inside a genuine TEE. Its identity is measured by a cryptographic hash of the memory pages of its code [97, 98]. This proof is signed by a secret key loaded inside the CPU when manufactured or remotely afterwards. This mechanism provides a way to verify that a computation result was produced by the expected code.

2.2.2 . Trust Model and Limitations

Trust. TEEs rely on a trust model where the processor's chip package is the Trusted Computing Base [93]. All the other hardware and software components are considered untrusted. An assumption of current TEEs' design is that the code loaded inside the secure environment is considered trusted as well. Indeed, using TEEs is not a silver bullet towards security: a vulnerable code loaded inside a TEE can still be abused to leak sensitive information [99]. Likewise, the code isolation is not both ways: a malicious code loaded inside the trusted environment can leak its potentially confidential data to the untrusted environment, encrypt documents or participate in denial-of-service attacks [100]. This has led to research works such as [101, 102, 103, 104] which propose solutions to isolate programs within their enclaves, a technique known as *sandboxing*.

Compatibility. While TEEs' security properties are appealing, using them requires a particular programming model. Indeed, the capabilities of isolated codes are limited, one of the foremost limit being the absence of system calls inside the trusted environment. As such, a program leveraging TEEs must be structured with, on the one hand a fully-featured code running on the untrusted execution environment, and on the other end a set of trusted functions which has to rely on this untrusted part to carry out system calls but must proceed to verification of their results for the sake of security. Consequently, proposals have been made to ease the porting of existing applications to TEEs. For example, [105, 106,

107, 108] enable unmodified applications to be run on Intel SGX by considering the entire application as trusted and offering an interface to handle system calls. Others [109, 110] have proposed help in porting existing applications via source code annotations or binary post-processing. Finally, [111, 112, 113, 114] help in the development of new applications leveraging TEEs.

Performances. Depending on the application, relying on TEEs can have a significant impact on performances. The RAM being encrypted, every memory access has to perform a decryption or encryption for reads and writes, as well as maintaining the integrity through, for example, a Merkle Hash Tree [93]. Also, in order to enter and leave the trusted environment, for example to operate a system call, the processor has to perform a costly context-switch [105, 115] involving security verifications and cache flush. Research proposals [105, 116] were aimed at limiting this cost by offering solutions to answer systems calls without leaving the trusted environment. Finally, on Intel SGX the portion of the RAM allocated for all enclaves and integrity measurements –called the Processor Reserved Memory– is limited to 128MB or 256 MB [117][1]. When this memory is full, a costly paging mechanism is used [116, 119], evicting its pages while ensuring their confidentially, integrity and freshness.

Side-channels. On another hand, the security of TEEs is not foolproof and has been the subject of many research works. A first class of side-channel attacks have been shown to exploit vulnerabilities of the inner mechanisms of TEEs, including the branch predictions [120, 121] and the cache evictions [122] to break the data confidentiality. Even worse, some [121, 123] even succeed in extracting the attestation key of the TEE, enabling the forgery of attestation for fake enclaves. This class of attacks is typically addressed by microcode updates from TEE providers. Another class of side-channel attacks is inherently linked to the architecture design of TEEs. Because the trusted environment is executed alongside the untrusted one on the same machine and rely on it for various purposes (e.g. allocating memory pages), information can be extracted by the untrusted environment. Attacks exploiting cache timing [124, 125, 126], access patterns [127] or the paging mechanism [128] have been seen to reveal the data manipulated by programs running inside TEEs. Consequently, research works have been carried out to try to mitigate cache timing attacks [129, 130, 131, 132], the exploitation of the paging mechanism [132, 133] and access patterns [134, 135].

2.2.3 . Conclusion

Trusted execution environments are a ubiquitous technology bringing strong security properties on end-user devices that were until then deemed unfit to carry out computations without the extreme assumption of trusting the entire software and hardware stacks. Their properties of *code isolation*, *data protection* and *at-*

[1]This limit has been recently increased to up to 1 TB in the last generation of Intel XEON Processors [118].

testation offer a way to execute code without being influenced or eavesdropped by the rest of the system, including the privileged operating system and hypervisor.

The particular programming and trust models of TEEs entail an adaptation of existing systems to be compatible. Nonetheless, their ubiquity has contributed to their inclusion in many different fields of work such as distributed computations [136, 137, 138, 139, 140], network traffic management [141], password managers [142], industrial control systems [143], deep neural networks [144], federated learning [145] and web-application scalability [146] to only cite a few. The following section will discuss their use for database management systems.

In this thesis, we rely on TEEs and sandboxing techniques to protect the personal data management of the user on their untrusted device (See Chapter 3). While the protections provided by TEEs have been shown to have weaknesses through side-channel attacks, their mitigation is considered out-of-scope and orthogonal to our work.

2.3 . DBMSs and Secure Hardware.

A field of application for TEEs which is of particular interest to this thesis is database management systems (DBMSs). As TEEs and other secure hardware bring security, their value for DBMSs can be significant. Indeed, by storing vast amount of data the security of these systems is a crucial factor, especially as they are often outsourced to third parties. Being able to outsource a database while maintaining security of the data with complete and efficient query-processing would be ideal. This section gives an overview of different adaptations of secure hardware to DBMSs (see [147] for a recent tutorial on this matter).

2.3.1 . Secure Hardware

Before the advent of TEEs, proposals where data management systems are coupled with secure hardware were already discussed. As introduced by Section 2.1.6, embedding an entire DBMS on secure chips can provide strong security properties. Spanning a fifteen years period since the early 2000s, works including smartcards or secure microcontrollers with DBMSs have been proposed with specific database designs to cope with the severe hardware limitations of these technologies [69, 70, 71, 72, 75, 84]. Due to the inner constraints of the concerned hardware, these solutions still suffer from poor performances and cannot be easily extended to support a large panel of computations.

Afterwards, other proposals have been made trying to tackle these issues. TrustedDB [148] proposed a DBMS ensuring data confidentiality via encryption. The data is encrypted at-rest and only decrypted on a tamper-proof trusted hardware during query execution. The query is divided between i) operators applied to only non-encrypted columns which are run by MySQL on commodity hardware and ii) operators including encrypted columns which leverage a secure co-processor running a modified version of SQLite. Cipherbase [149] partly reuses the same idea

by placing a field-programmable gate array (FPGA) inside an untrusted machine. Instead of employing two DBMSs, they modified one and run only the necessary primitives on the FPGA to minimize the performance decrease it incurs. Cipherbase notably consider access patterns attack and design oblivious implementations of scan-based relational operators. These two proposals bring rich functionalities similar to a fully-featured DBMS with better performances than cryptographic-based solutions. In their highest settings, they offer strong data confidentiality guarantees. However, they require specialized secure hardware (secure co-processor or FPGA) which hinders their availability.

2.3.2 . Advent of TEEs

With TEEs being embedded inside many different devices, their ubiquity naturally draw the attention of the database community as a potential tool to bring security to data management systems. Another main advantage is that their performances are orders of magnitude faster than smartcards or secure microcontroller.

CryptSQLite [150] proposes to pair an SQLite engine with an SGX enclave. The engine is loaded entirely inside the enclave while the data are stored encrypted outside of the enclave with a symmetric encryption scheme. Remote users undergo an initialisation phase where they exchange a symmetric key with the enclave. Then, they send encrypted queries to load and retrieve data. Data integrity and freshness is not considered. In EnclaveDB [151] the authors propose to embed a Hekaton query-processing engine and all sensitive data such as tables, indexes, and queries inside Intel SGX. Their proposal provide confidentiality, integrity and freshness for data and queries. In particular, they designed a protocol to check the integrity and freshness of the database log, allowing a secure database recovery. Compared with a classical DBMS running on non-secure platform, the overhead is up to 40% in their experiments. StealthDB [152] uses a slightly modified PostgreSQL with Intel SGX to ensure the confidentiality of the data through data encryption and obliviousness of database operators. Most of the DBMS is executed outside of SGX, the enclave is only used during query execution to perform primitive operators over encrypted data items. ObliDB [135] focuses on hiding the access patterns of an SQL engine. Unlike Cipherbase, they support efficiently oblivious point and small range queries using SGX.

Instead of ensuring the confidentiality of data, VeriDB [153] focuses on the problem verifiability of Select-Project-Join-Aggregate queries. By using a query engine inside an SGX enclave, their proposal provides a way to verify the correctness and completeness of the data during query execution. The overhead for analytical workloads is 9% to 39%.

Industrial uses of TEEs for DBMSs can be find in Azure SQL [154] which allows for encrypted column processing within an enclave, with the cryptographic keys owned client-side being passed to the enclave at runtime, to ensure data confidentiality. EdgelessDB [155] is an open-source project which can be deployed with docker and employs SGX to secure a MariaDB SQL engine.

Also noteworthy, other proposals aim at adapting indexing methods to TEEs, which are essential elements of database systems. HardIDX [156] proposes a secure and efficient B-tree indexing using SGX. Using this TEE, they achieve several orders of magnitude lower query processing time compared with the best known searchable encryption schemes and formally proved to be similar in the level of confidentiality. Because the access patterns of indexing systems can leak information, Oblix [157] provide an indexing method coupled with oblivious accesses. A particularity of Oblix is that their implementation is doubly-oblivious: the access patterns to both the internal memory of the enclave and the external memory are oblivious.

Finally, IronSafe [158] proposes an interesting design leveraging two different TEEs (Intel SGX and ARM TrustZone) to provide confidentiality, integrity and freshness of data. They implement Near Data Processing benefiting from the inclusion of low-power ARM CPUs inside storage devices paired with a more powerful host with Intel SGX to enable highly efficient and secure SQL query processing on computing storage architectures.

2.3.3 . Conclusion

This section described many adaptations of database management systems to secure hardware, showing the broad interest of this technology by the database community. While these proposals achieve various security properties like data confidentiality and potentially integrity/freshness with respect to the considered threat models, they all assume that the data processing code is trusted by the data owner. Support for User-Defined Functions/User-Defined Aggregate functions is not explicitly mentioned, but would imply the same trust assumption: the UDF code has to be trusted to be run on the DBMS. This thesis focuses on another problem not covered by current works: we make an assumption of untrusted third-party function code for the data owner and propose solutions to limit data leakage caused by this untrusted code.

2.4 . User-Defined Functions

While DBMSs are a powerful tool to manage data, the set of functions included by their editors is limited and an extension of this set implies an upgrade of the software by the editor and on the users' machine. Because DBMS editors cannot include every possible functions that users will need, a mechanism to support custom-made functions was rapidly needed. User-Defined Functions (UDFs) are an elementary concept of DBMSs, being introduced in the very early age of relational databases [159, 160] as a way to provide *extensibility* to these systems. Since their introduction, UDFs are still a feature of major commercial DBMSs [161, 162, 163]. Depending on the DBMS, users can write functions in programming languages such as PL/SQL, Java, or Python, bind them to the DBMS and then use them inside queries in the same way as built-in functions. With this in mind, the concept of UDFs is a solution to bring extensibility to data management. Our work involves

untrusted UDFs, thus we are particularly interested in existing works regarding the security of UDFs and this section brings an overview of this subject.

Existing works target different aspects regarding the *security* of UDFs: server security, code security or data security.

2.4.1 . Ensuring Server Security

Because their code can be prone to bugs or vulnerabilities, one of the first security measure to be implemented was to dissociate UDFs from the main DBMS process. [164] illustrates this by placing UDFs inside a Java Virtual Machine which prevents harmful I/O of UDFs as well as restricting their namespace and preventing them from affecting threads of other UDFs. The goal here is to protect the DBMS integrity and ensure its stability. More recent works such as [165] isolates the code via virtualization (modified hypervisor). UDFs in DBMSs are one application domain of [165] but also untrusted library usage or serverless contexts.

2.4.2 . Ensuring Code Security

A user might be reluctant to share the code of their UDF because it might be a valued asset or is subject to intellectual property. In this case, ensuring the security of the UDF means being able to compute queries involving this UDF without having to bind the code of the UDF on a server used by multiple users. An example can be found in [166] which leverages DMBS clients to compute such UDF and minimizes the performance overhead.

2.4.3 . Ensuring Data Security

Snowflake Secure UDF [167] or Google BigQuery Authorized Functions [168] bring a solution to the case where a user u_1 who possesses data wants to authorize another user u_2 to execute a UDF on these data without having access to said data. These solutions notably disable some features of the query optimizer to avoid leaking data (e.g., no selection pushed down the execution tree to avoid revealing certain input data). In these cases, the UDF code as well as the server running it are trusted by the data owner u_1.

Similarly to UDFs in DBMS, data security in the Function-as-a-Service (FaaS) model has also been the subject of research works. In this model, also called the serverless paradigm [169], cloud users can submit computations at the granularity of a function according to their needs. Unlike Infrastructure-as-a-Service or Platform-as-a-Service, FaaS ease the burden of architecture or server management for the cloud user, delegating this task to the cloud provider. Again, because there is a delegation of computation, the security of the computed data has to satisfy the clients' needs. In [169], the authors propose to leverage TEEs like Intel SGX to protect the data computed by those function from an untrusted cloud provider. Additionally, they rely on the isolation of a Javascript Engine to isolate one function from the others, all running inside the same SGX enclave. Contrarily, [170] executes each function inside its own SGX enclave.

2.4.4 . Conclusion

While the security of UDFs has been studied in the past, the related works in this matter focus on different issues than the one faced by this thesis: they either lessen the impact on the DBMS stability of buggy or vulnerable UDF or keep secret their code. Other works which target data security either aim at allowing an untrusted user to run a trusted UDF, or prevent an untrusted cloud provider from reading the sensible data processed by the UDF. This thesis studies another problem: running a UDF that is untrusted by the data owner. Particularly, we consider that the UDF is sufficiently isolated to ensure the security of the data it manipulates against an untrusted platform and against voluntary leakage through channels other than the computation result. We want to limit the amount of data that can be leaked through this particular computation result. Our architecture is detailed in Chapter 3.

2.5 . Alternative Privacy Approaches and Positioning

Preserving the privacy of users by supporting data confidentiality but still enabling advanced and generic computations on these data is a prevalent issue crossing together different concepts of computer security and data management. This section introduces related works on this matter.

2.5.1 . Homomorphic Encryption

Homomorphism is a property of some cryptography systems which enables computations to be carried out directly on the ciphertext instead of having to decrypt it beforehand. This is a desirable property as it preserves data confidentiality. Homomorphism were first described in [171]. The idea is as followed: Given a cryptosystem where E is the encryption function and D the decryption function, this system is said to be homomorphic if $E(m_1) \oplus E(m_2) = E(m_1 \odot m_2)$ where m_1, m_2 are messages and $\oplus, \odot$ are mathematical operations (which can be equal in certain cases). Three categories of homomorphic encryption exist:

- **Partially homomorphic** cryptosystems which support only one operation on cyphertexts. For example, the RSA cryptosystem [172] is partially homomorphic for the multiplication.
- **Somewhat homomorphic** encryption which supports both the addition and multiplication on cyphertexts but only for a given number of times. For example, [173] allows only one multiplication but an unlimited number of additions.
- **Fully homomorphic** cryptosystems which handles both the addition and multiplication for an unlimited number of times, thus being generic in the types of computations supported on encrypted data. The first fully homomorphic proposal was made in 2009 [174].

In our context, CryptDB [175] is an interesting application of homomorphic encryption to a database management system. It leverages partially homomorphic encryption schemes to support a subset of database operators.

Regarding the studied problem in this thesis, partially and somewhat homomorphic encryption do not meet our requirements as we aim for rich and diverse computations, which is the basis of our extensibility need. Fully homomorphic encryption, although very promising, are not yet practical in terms of performances: encrypting 128 bits of data costs tens of seconds [176].

2.5.2 . Local Differential Privacy

Another way of computing users' data without hurting their privacy is to employ anonymization techniques on the data beforehand. The data is "distorted" in a special way to avoid the recovery of personal information while still allowing computations to be carried out. Proposals such as k-anonymity [177], l-diversity [178], t-closeness [179] and ϵ-differential privacy [180] offer solutions to anonymize data sets containing the data of several individuals in order to be able to perform computations on such data sets without hurting the privacy of participating users. These solutions focus on the anonymization of data pertaining to multiple users, intending to conceal each user into the masses, and as such cannot be applied to our context where the data of a single individual is computed. However, a subsequent work called Local Differential Privacy deserves to be discussed.

The general idea behind differential privacy [180] can be roughly summarized as: *Given two data sets D_1 and D_2 which differs by only one tuple, the result of any computation on each data set should differ by no more than ϵ* (assumed small). Originally, differential privacy needs a trusted entity which gathers the data of all users and answers differentially private results. This strong trust assumption led to the creation of Local Differential Privacy [181] where users can distort their data themselves before revealing them.

The origin of Local Differential Privacy is the randomized response technique [182], which proposed a way to protect respondents of a survey by employing randomness. Before answering a question, the respondent secretly tosses a coin. If the coin falls on heads, the respondent answers truthfully. Otherwise, they toss a second coin and answers "No" for heads and "Yes" for tails. The respondents will give the true answer 75% of the time while still having plausible deniability for their answers. The operation thus added noise to results.

While some real-world applications of differential privacy have been implemented [183, 184], it is hard to design algorithms with both acceptable utility and acceptable level of privacy. Also, because it adds noise to the result, the accuracy cannot be preserved. In both our Energy and Green Bonus examples this is a major issue as the result have consequences on a financial decision. In the end, local differential privacy is reserved for applications involving many participants to compensate for the added noise and keep a sufficient utility of the data. Thus, Local Differential Privacy is unsuitable in our context.

2.5.3 . Information Flow Analysis

In order to prevent information leakage of a given code, several solutions revolving around the concept of Information Flow [185] have been proposed. Roughly described, when considering a computer program, Information Flow studies how does information spread inside the code of this program, from its input to its output. This flow of information is regulated by a confidentiality policy, which can be either defined by the code programmer using type annotations on variables [186] and enforced by the compiler/interpreter or verified automatically by a source code or binary analyzer [187]. Some research works [185] focus on the enforcement of a strong confidentiality policy called noninterference policy. This policy assumes that the computation will involve confidential data, and that no public outputs of the computation program should reveal confidential data, even partially. More precisely, "a variation of confidential input does not cause a variation of public output" [185]. In our context, the noninterference policy is not applicable since it is legitimate –and an intrinsic goal of computing an aggregation function– for the third-party code to compute outputs that depend on sensitive inputs.

In [187], the authors argue that the noninterference policy is too strong for machine learning algorithms and is always violated in this context. Thus, they define a new policy called nonreversibility policy as "a variation of a single confidential input could cause a variation of public outputs, but keeping this single confidential input and non-confidential inputs unchanged will not always lead to the same public outputs". Noticeably, they consider the case of an untrusted machine learning algorithm running inside an SGX enclave. This proposal has some limitations in our context. First, it requires the availability of the source code of a given application needing validation. Second, their prototype adds 1,000 lines-of-code (LOC) on top of Clang 7.0.0 and it is unclear if the entire Clang compiler has to be trusted. Finally, it is also unclear how their proposal impacts computations other than machine learning algorithms.

Other works such as [188, 189] leverage code analysis, another information flow technique. Their solution is tailored to the smartphone context where they study the behavior of applications at runtime by labelling the data from confidential sources and analysing their propagation through program variables, messages between applications, system library calls and file reads/writes. These solutions are limited in the number of leakage channels they can cover. For example, they do not cover implicit data flows, e.g., when the content of a given variable is directly dependent on a confidential data without being a direct assignment. Also, their solution only supports the discovery of malicious app a posteriori, through a logging system. Finally, they assume trusted an entire virtual machine and native system libraries.

2.5.4 . Leakage Prevention through Traffic Analysis

Other solutions to prevent information leakage, also exemplified in the context of smartphone applications, have been proposed. Proposals such as [190, 191] limit the leakage of Personally Identifiable Information (PII) accessible by applications such as the phone number, the device IMEI, GPS coordinates or the user's name. These proposals rely on analysing the internet traffic of the device to find network packets containing such information and block their sending. In [190], the authors proposed a string-matching algorithm executed directly on the user's device to find PII in large bodies of text and propose a choice to the user: either allow or refuse the sending of the containing packet. The algorithm is designed to detect the presence of a PII even if it has been slightly modified (characters insertions, deletions or substitutions). They also propose an automatic classifier to predict the user's choice based on past decisions, avoiding too many user interactions. [191] leverages a server which intercepts all packets from devices and search for PII via a machine-learning enhanced algorithm.

Although these solutions offer a way to control data leakage of potentially malicious code, they do not meet our requirements in our context. Indeed, they cannot prevent leakage of PII that have been heavily modified, for example via encryption, custom-made encoding or compression.

2.5.5 . Inference Attacks

It is important to distinguish the studied problem in this thesis from the inference attacks which have been analyzed, for example, in the context of statistical databases [192]. Even if the third-party UDF code used to compute the user's data is untrusted, in the best case scenario it complies strictly with the task it is supposed to perform. In our Energy example, it means that the function installed by the customer strictly computes, for example, the sum of energy they consumed over a time period. Even then, the result is subject to inference attacks. Because aggregates are not functions which rigorously hide the input data of a given result, it is possible to infer data considering only the results of computations. In its most basic form, the idea behind inference attacks can be illustrated by the following example: Consider two queries $Q_1 =$ *SELECT SUM(consumption) FROM ElecTraces WHERE date between '2022-01-01 00:00' AND '2022-01-31 23:59:59'* and $Q_2 =$ *SELECT SUM(consumption) FROM ElecTraces WHERE date between '2022-01-01 00:00' AND '2022-01-31 22:59:59'*. An attacker receiving the results of both queries can infer exactly how much energy was consumed on January 31, 2022, between 23:00 and 23:59.

Solutions for this problem have been proposed in the past, especially in the domain of statistical databases, and have been evaluated in [192]. They are divided into four categories: Conceptuals [193], query restrictions [194], statistical perturbation of input data [195] and perturbation of results [196].

In this thesis, we do not consider inference attacks on correct results. Because

the computing code is untrusted and written by third parties, it can perform a more devastating attack by including raw data directly inside the result instead of the statistics it is supposed to compute, and we want to protect against this type of attacks.

3 - Problem Formulation

Contents

In this chapter, we formalize the problem addressed in this thesis. First, we describe the Extensive and Secure PDMS architecture that we consider and its associated security properties. Then, we detail our computing model focusing on a common type of data processing on a PDMS, namely *aggregate functions*. We study a data leakage problem, emanating from a powerful attacker programming the data processing code, explained in the subsequent threat model. Finally, we formulate the exact questions that we want to answer and define the required metrics.

3.1 . Extensive and Secure PDMS Architecture

3.1.1 . Architecture Outline

Designing a PDMS architecture that offers both security and extensiveness is a significant challenge: security requires trusted code and environment to manipulate data, while extensiveness relies on user-defined and thus potentially untrusted code. In a previous publication [23], the PETRUS team proposed a logical (i.e., abstract) three-layers architecture to handle this tension, where *Applications* (Apps) on which no security assumption is made, communicate with a *Secure Core* (Core) implementing basic operations on personal data, extended with *Data Tasks* (Data tasks) isolated from the Core and running user-defined code (see Fig. 3.1).

Core. The Core is a secure subsystem that is a Trusted Computing Base (TCB). It constitutes the sole entry/exit point to manipulate PDMS data and

retrieve results, and hence provides the basic DBMS operations required to ensure data confidentiality, integrity, and resiliency. It therefore implements a data storage module, a policy enforcement module to control access to PDMS data, a basic query module (as needed to evaluate simple selection predicates on database objects' metadata) and a communication manager for securing data exchanges between the architecture layers and with the Apps. As a TCB, the Core's capabilities should be cut down at the essentials to limit the potential vulnerabilities and make audits easier.

Data tasks. Data tasks can execute arbitrary, application-specific data management code, thus enabling extensiveness (like UDFs in traditional DBMSs). The idea is to handle UDFs by (1) dissociating them from the Core into one or several Data tasks evaluated in a sufficiently isolated environment to maintain control on the data sent to them by the Core during computations, and (2) scheduling the execution of Data tasks by the Core such that security is globally enforced.

Apps. The complexity of applications (large code base) and their execution environment (e.g., web browser) make them vulnerable and potentially malicious. Therefore, no security assumption is made on them. They have no privilege on raw data and only manipulate data resulting from Data tasks and authorized by the Core.

Logical versus physical architecture. The three layers of this logical architecture can be implemented into different physical architectures depending on the use cases. The straightforward implementation would be to run the Core and the Data Tasks on the same machine, being on a user-device or on a cloud PDMS (See Section 2.1 for an overview of existing solutions). However, other cases where they are not run alongside can be envisioned: For example, a user of a self-hosted PDMS can be inclined to rely on remote Data Tasks hosted on a powerful server if its own device is not powerful enough to handle some tasks, or because the code of the UDF is protected by intellectual property [101].

3.1.2 . Security Properties

The specificity of our architecture is to remove from local or remote Apps any sensitive data-oriented computation, delegating it to Data tasks running UDFs under the control of the Core. Each App requiring the use of UDFs leverages an App Manifest which includes essential information about the UDFs it needs. This App Manifest should contain:

- The purpose of the UDFs.
- The set of PDMS objects needed to be accessed by the UDFs.
- The size of the result produced by each UDF.
- The identities (i.e., code measurements) of the Data Tasks carrying out the UDF's computation.

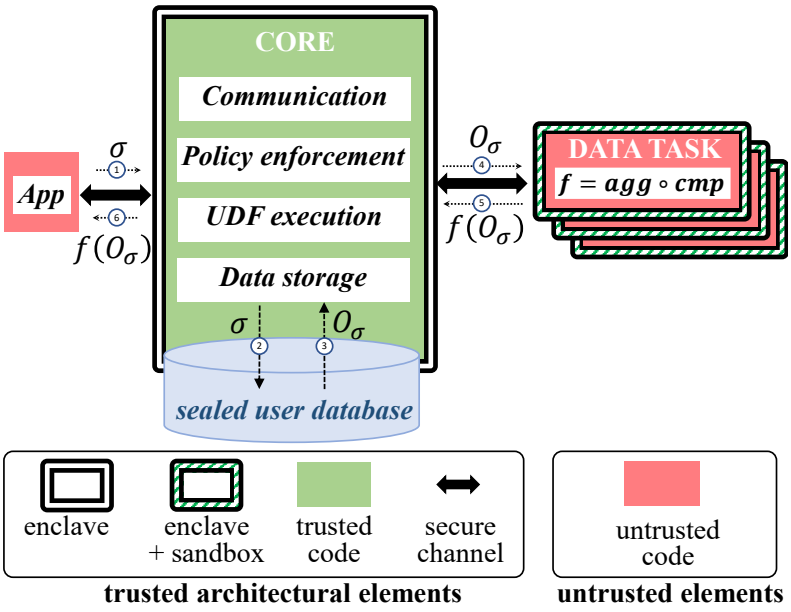


Figure 3.1: Architecture overview.

- For UDFs requiring the use of multiple Data Tasks, the order in which they should be executed.

The manifest should be validated, e.g., by a regulatory agency or the App marketplace, and approved by the PDMS user at install time before the App can call corresponding functions. Specifically, our system implements the following architectural security properties:

P1. Enclaved Core/Data tasks. The Core and each Data task run in *individual enclaves* protecting the confidentiality and the integrity of the data manipulated as well as the integrity of the code execution against the untrusted execution environment (application stack, operating system). Such properties are provided by TEEs, e.g., Intel SGX, which guarantees that (i) enclave code cannot be influenced by another process; (ii) enclave data is protected from any other process (RAM encryption); (iii) enclave can prove that its results were produced by its genuine code [93]. Besides, the code of each Data task is sandboxed [101] within its enclave to preclude any voluntary data leakage outside the enclave by a malicious Data task function code. Such Data task *containment* can be obtained using existing software such as Ryoan[101] or SGXJail[103] which provide means to restrict enclave operations (bounded address space and filtered syscalls).

P2. Secure communications. All the communications between the Core, the Data tasks and the Apps are classically secured using TLS to ensure authenticity, integrity and confidentiality. Because the inter-enclave communication channels are secure and attested, the Core can guarantee to the Apps the integrity of the UDFs being called.

P3. Core as proxy. In the specific case of data collection Tasks requiring authentication to web services, the Core also plays the role of a proxy opening the secure communication channel in lieu of the Data Task. This way the *connection credentials are fully protected* since this information is never revealed to the collect Data Tasks. This ensures that the Core is the only entry/exit point of the PDMS, while Data Tasks never have a direct access outside of their enclave.

P4. End-to-end access control. The input/output of the Data tasks are regulated by the Core which enforces *access control* rules for each UDF required by an App. Also, the Core can apply basic selection predicates to select the subset of database objects for a given UDF call. For instance, in our 'Green bonus' example, a Data task running a UDF computing the number of bike commutes during a certain time interval will only be supplied by the Core with the necessary GPS traces (i.e., covering the given time interval). If several Data tasks are involved in the evaluation of a UDF, the Core guarantees the transmission of intermediate results between them. Finally, the App only receives final computation results from the Core (e.g., number of bike commutes) without being able to access any other data.

P5. End-to-end attestations. Altogether, the previous security properties result in an end-to-end attestation ensuring a third party the verifiability of a computation. In our example, this is done in three steps. First, P2 and P4 provide integrity for the GPS traces given to the data collection Task and stored by the Core (e.g., data is protected against unauthorized updates from the PDMS owner). Second, P1 and P4 certify that the final result (e.g., the number of bike trips) is produced by an isolated enclave (the computation task) acknowledged by the Core. Finally, the Core guarantees that the global execution plan complies with the App manifest.

The above properties enforce the PDMS security and, in particular, the data confidentiality, precluding any data leakage, except through the legitimate results delivered to the Apps.

3.2 . Computation and Threat Models

3.2.1 . Computing Model

We seek to propose a computation model for UDFs (defined by any external App) that satisfies the canonical use of PDMS computations (including the use-cases discussed in Chapter 1). The model should not impact the application usages, while allowing to address the legitimate privacy concerns of the PDMS users. Hence, we exclude from the outset UDFs which are permitted by construction to extract large sets of raw database objects (like SQL select-project-join queries). Instead, we consider UDFs (i.e., a function f) as follows: (1) f has legitimate access to large sets of user objects and (2) f is authorized to produce various results of fixed and small size.

The above conditions are valid, for instance, for any aggregate UDF applied

to sets of PDMS objects. As our running example illustrates, such functions are natural in the PDMS context, e.g., leveraging user GPS traces for statistics of physical activities, the traveled distance or the used modes of transportation over given time periods.

Aggregates in a PDMS are generally applied on complex objects (e.g., time series, GPS traces, documents, images), which require adapted and advanced UDFs at the object level. Specifically, to evaluate an *aggregate* function agg, a first *computation* function cmp processes each object o of the input. For instance, cmp can compute the integral of a time series indicating the electricity consumption or the length of a GPS trace stored in o, while agg can be a typical aggregate function applied subsequently on the set of cmp results. Besides, we consider that the result of cmp over any object o has a fixed size in bits denoted by $||cmp||$ with $||cmp|| << ||o||$ (e.g., in the examples above about time series and GPS traces, cmp returns a single value –of small size– computed from the data series –of much larger size– stored in o). For simplicity and without lack of generality, we focus in the rest of this thesis on a single App a and computation function f.

Figure 3.1 shows an example of a computation. First, the App requests the execution of f to the Core including a predicate σ. Second, the Core applies σ to select a subset O_σ of the data authorized to the App according to its Manifest (previously validated). Then, the Core demands the creation of a Data Task containing the code of f and establishes a secure communication channel with it, simultaneously verifying its identity. Afterwards, the Core sends the input set O_σ to the Data Task for computation and receives the result $f(O_\sigma)$. Finally, it forwards the result to the App.

Computing model. An App a is granted execution privilege on an aggregate UDF $f = agg \circ cmp$ with read access to (any subset of) a set O of database objects according to a predefined App manifest $\{< a, f, O >\}$ accepted by the PDMS user when the App is installed. a can freely invoke f on any $O_\sigma \subseteq O$, where σ is a selection predicate on objects metadata (e.g., a time interval) chosen at query time by a. Function f computes $agg(\{cmp(o) \mid o \in O_\sigma\})$, with cmp an arbitrarily complex preprocessing applied on each object $o \in O_\sigma$ and agg an aggregate function. We consider that agg and cmp are deterministic functions and produce fixed-size results.

3.2.2 . Threat Model

We consider that the attacker cannot influence the consent of the PDMS user, required to install UDFs. However, neither the UDF code nor the results it produces can be guaranteed to meet the user's consented purpose. To cover the most problematic cases for the PDMS user, we consider an active attacker fully controlling the App a with execution granted on the UDF f. Thus, the attacker can authenticate to the Core on behalf of a, trigger successively the evaluation of f, set the predicate σ defining $O_\sigma \subseteq O$, the input set, and access all the results produced by f. Furthermore, since a also provides the PDMS user with the code of

$f = agg \circ cmp$, we consider that the attacker can instrument the code of agg and cmp such that instead of being deterministic and producing the expected results, the execution of f produces some information coveted by the attacker, in order to reconstruct subsets of raw database objects used as input.

On the contrary, we assume that security properties P1 to P5 (see Section 3.1.2) are enforced. In particular, we assume that the PDMS Core code is fully trusted as well as the security provided by the TEE (e.g., Intel SGX) and the in-enclave sandboxing technique. Fig. 3.1 illustrates the trust assumptions on each of the architectural elements of the PDMS involved in the evaluation.

Note that to foster usage, we do not impose any restrictions on the σ predicates nor on the App query budget (see Section 8.2). In addition, we do not consider any semantic analysis or auditing of the code of f since this is not realistic in our context as the layman PDMS user cannot handle such tasks and the Core, being our Trusted Computing Base, should be kept minimal. These solutions are also mostly complementary to our work as discussed in Chapter 2.

3.3 . Problem Formulation

3.3.1 . Research Questions

The precise goal of the thesis is to address the following two questions:
(1) Is there an upper bound on the potential leakage of personal information that can be guaranteed to the PDMS user, when evaluating a user-defined function f successively on large sensitive data sets, under the considered PDMS architecture, computation and threat models?
(2) Is there a performance-acceptable execution strategy guaranteeing minimal leakage with potentially large volumes of personal data?

Answering these questions is critical to bolster the PDMS paradigm. A positive conclusion to the first question is necessary to justify a founding principle for the PDMS, insofar as bringing the computational function to the data (and not the other way around) would indeed provide a quantifiable privacy benefit to PDMS users. The second question may lead to a positive assessment of the realism and practical adoption of the proposed solutions.

Analysis of the problem requires appropriate quantification of leakage for attacks conducted using corrupted code, within the framework of the computational and threat models described earlier. Before presenting our simplified metrics, let us consider a simple attack example.

3.3.2 . Attack Example

Attack example. The code for f, instead of the expected purpose which users consent to (e.g., computing bike commutes for the 'Green Bonus'), implements a function f_{leak} that produces a result called $leak$ of size $||f||$ bits ($||f||$ is the number of bits allowed for legitimate results of f), as follows: (i) f sorts its input object set O, (ii) it encodes on $||f||$ bits the information contained in O next to the previously leaked information and considers them as the –new– $leak$; (iii) it sends –new– $leak$ as the result.

In a basic approach where the code of f is successively evaluated by a single Data Task DT^f receiving the same set O of database objects as input, the attacker obtains after each execution a new chunk of information about O encoded on $||f||$ bits. The attacker could thus reconstruct O entirely by assembling the received $leak$ after at least $n = \frac{||O||}{||f||}$ successive executions, with $||O||$ the size in bits needed to encode the information of O.

3.3.3 . Leakage Metrics

To address the formulated problem, we introduce metrics to quantify the amount of data received by the attacker. We denote by $||x||$ the amount of information in x, measured by the number of bits needed to encode it. For simplicity, if x is a function we consider this value as the size in bits of the result produced by this function. If x is a database object or set of objects, we consider its footprint in bits. We define the leakage $L_f(O)$ resulting from successive executions of a function f on subsets of O allowed to f, as follows:

Definition 1. Data set leakage. The successive executions of f, taking as input successive subsets O_σ of a set O of database objects, can leak up to the sum of the leaks generated by non identical executions of f. Two executions are considered identical if they produce the same result on the same input (it is the case for functions assumed deterministic). Thus, n successive non-identical executions can generate up to a *Data set leakage* of size ${L_f}^n(O) = ||f|| \times n$ (i.e., each execution of f provides up to $||f||$ new bits of information about O).

To quantify the number of executions required to leak given amounts of information, we introduce the *leakage rate* as the ratio of the leakage on a number n of executions, i.e., $\overline{{L_f}^n} = \frac{1}{n} \cdot {L_f}^n(O)$.

The above leakage metrics express the 'quantitative' aspect of the attack. However, attackers could also focus their attack on a (small) subset of objects in O that they consider more interesting and leak those objects first, and hence optimize the use of the possible amount of information leakage. To capture the 'qualitative' aspect of an attack, we introduce a second leakage metric:

Definition 2. Object leakage. For a given object o, the *Object leakage* denoted ${L_f}^n(o)$ is the total amount of bits of information about o that can be obtained after executing n times the function f.

3.3.4 . Conclusion

The research questions presented above can be formulated as the following problem statement:

1. Identifying the means leveraged by an attacker to leak data will enable the conception of appropriate countermeasures to prevent them from using these means. These countermeasures have to be compatible with our architecture, computation and threat models while being effective enough to specify an upper bound on data leakage. This upper bound and the impact of the countermeasures have to be evaluated using the predefined leakage metrics.

2. In order to provide efficiency, it is necessary to identify the constraints and performance bottlenecks of our architecture. Then, general computation execution plans must be implemented coping with these constraints while implementing the leakage countermeasures.

As such, we devise two important steps in the following chapters of this thesis. First, we propose three security design rules to be enforced during the computation of f to limit the leakage of data measured by our leakage metrics defined above. Named *security building blocks*, they enable the definition of an upper bound on said leakage which can be brought down to a minimum. They are the focus of Chapter 4. Second, we build upon these design rules and provide algorithms to efficiently enforce this upper bound, named *execution strategies*. They are the contributions of Chapter 5.

4 - Leakage Control - Security Building Blocks

Contents

This chapter introduces three security building blocks previously sketched in [24] to control the potential data leakage occurring over successive computation results produced by a UDF $f = agg \circ cmp$, executed inside a Data task DT^f, on a set of objects O. These building blocks allow the establishment of an upper-bound on the amount of data that can be leaked through the results of computations.

4.1 . Stateless Data Tasks

Since the potential attacker controls the code of f, an important lever that can be exploited is data persistency, as keeping a *state* between successive executions maximizes leakage. For instance, in the Attack example (Section 3.3.2), f maintains a variable $leak$ according to previous executions to avoid leaking same data twice. Persistent states can be exploited by f –although executed as Data task DT^f– without hurting the security hypotheses, e.g., by leveraging in-memory storage or resorting to PDMS database or secure file system.

A first building block to mitigate this lever is to rely on *stateless* Data tasks with the objective of limiting the leakage rate in successive executions. This does not negatively impact usage as computation requests should be evaluated independently i.e. without being influenced by past or concurrent computations, comparable to database queries in standard DBMSs.

Definition 3. Stateless Data task. A stateless Data task is instantiated for the sole purpose of answering a specific function call/query, after which it is terminated and its memory wiped.

Enforcement. On SGX, statelessness can be achieved by destroying the Data task's enclave. It also requires to extend containment (security property *P1*) by preventing variable persistency between executions or direct calls to stable storage (e.g., SGX protected file system library). Obviously, PDMS database accesses must also be regulated by the Core.

Impact on leakage. Without the possibility to rely on a persistent state, a corrupted computation code running as a Stateless Data task may leverage randomness to maximize the leakage rate. For instance, in the Attack example, at

each execution, f_{leak} can select a random fragment of O to produce a $leak$, the randomness being drawn from APIs or timer/date. Even if the same query is run twice on the same input, two different fragments of O can be leaked. Considering a uniform leakage function, the probability of producing a new $leak$ is proportional to the remaining amount of data –not leaked yet– present in the data task input O. That is, the probability is high (i.e., close to 1) when none or only a few fragments of O have been already leaked, and it slowly decreases to 0 when O has been (nearly) entirely leaked, which increases the necessary executions to leak large amounts of data.

4.2 . Deterministic Data Tasks

The stateless property forces attackers to (i) employ randomness in selecting data fragments to be leaked in a computation to maximize the leakage, and (ii) choose the leaked fragment necessarily in the current computation input (previous inputs cannot be memorized). To further reduce leakage and tackle attacks relying on randomness, we enforce a new restriction for Data tasks, i.e., determinism:

Definition 4. Deterministic Data task. A deterministic Data task necessarily produces the exact same result for the same function code executed on the same input, which precludes leakage accumulation in the case of identical executions (enforcing Def. 1).

Enforcement. Data task containment (security property *P1*) can be leveraged to enforce data task determinism by preventing access to any source of randomness, e.g., system calls to random APIs or timer/date. Virtual random APIs can easily be provided to preserve legitimate uses, e.g., the need for sampling, as long as they are "reproducible", e.g., the random numbers used are forged by the Core using a seed based on the function code f and its input set O, e.g., $seed = hash(f||O)$. The same inputs (i.e., same sets of database objects) must also be made identical between successive Data task execution by the Core (e.g., sorted before being passed to Data tasks). Clearly, to enforce determinism, the Data task must be stateless, as maintaining a state between executions provides a source of randomness to the Data task. Note that another way to enforce determinism is to store the previous results produced by the data tasks for any different input objects set, and reuse the stored result instead of recomputing (see Section 5.2).

Impact on leakage. With deterministic (and stateless) Data tasks, the only remaining source of randomness in-between computations is the Data task input (i.e., $O_\sigma \subseteq O$). The attacker has to provide a different selection predicate σ for each computation in hope of maximizing the leakage rate. Hence, the average leakage rate of deterministic Data tasks is upper bounded by that of stateless Data tasks. In theory, the number of different inputs of f being high (up to $2^{|O|}$, the number of subsets of O), an attacker can attain similar Data set leakage with deterministic Data tasks as with stateless ones.

4.3 . Decomposed Data Tasks

By changing the selection predicate σ (i.e., as needed to favor rich usage for Apps), attackers may leak new data with each new execution of f, regardless if Data tasks are stateless and deterministic. The attacker could also concentrate leakage (see Def. 2) on a specific object o by executing f on different input sets, each one containing the object o. To mitigate the attack vector represented by the selection predicate σ, we introduce a third building block based on decomposing the execution of $f = agg \circ cmp$ into a set of Data tasks. On the one side a Data task DT^{agg} executes the code of agg, and on the other side a set of Data tasks $\{DT_i^{cmp} \mid i \in [1, m]\}$ executes the code of cmp on the partitioned set O of objects authorised to App, each partition P_i being of maximum cardinality k. Each Data Task DT_i^{cmp} produces one result per object in its input partition P_i.

The goal of this decomposition is to limit the Object leakage since information about objects in a given partition P_i can only leak into the (at most) k results produced by DT_i^{cmp}. This parameter k is called *Leakage factor*, as it determines the number of intermediate results in which information about any given object o can be leaked. An important observation is that to enforce a leakage factor of k, the partitioning of O in partitions of size at most k has to be 'static', i.e., independent of the computation input O_σ, so that any object is always processed within the same $k-1$ other objects across executions, such that the stateless and deterministic Data task processing this partition always produces the same result set. This additional restriction for Data tasks is defined as follows:

Definition 5. Decomposed Data tasks. Let $P(O) = \{P_i \mid i \in [1, m]\}$ with $m = \left\lceil \frac{|O|}{k} \right\rceil$ be a *static* partitioning of the set of objects O, authorized to function $f = agg \circ cmp$, such that every partition P_i is of maximum cardinality $k > 0$ (k being fixed beforehand, e.g., at install time). A decomposed Data tasks execution of f over a set of objects $O_\sigma \subseteq O$, involves a set of Data tasks $\{DT_i^{cmp} \mid P_i \cap O_\sigma \neq \emptyset\}$, with each DT_i^{cmp} being a stateless and deterministic Data task executing cmp on a given partition P_i containing at least one object of O_σ. Each DT_i^{cmp} is provided P_i as input by the Core and produces a result set $res_i^{cmp} = \{cmp(o) \mid o \in P_i\}$ to the Core. The Core discards all the results corresponding to the objects $o \notin O_\sigma$, i.e., objects that are not part of the current computation but have been computed nonetheless since they belong to partitions containing objects that *are* part of the current computation. Ultimately, a stateless and deterministic Data task DT^{agg} is used to aggregate the union of the results sets part of the computation, i.e., $\{cmp(o) \mid o \in O_\sigma\}$, to produce the final result.

As an illustration, Figure 4.1 shows a decomposed Data task execution, on a static partitioning of O with a Leakage factor $k = 3$, evaluating f on three objects (in orange) matching predicate σ and present in two partitions, with two Data tasks allocated to evaluate cmp on each partition, and one Data task evaluating agg on the result of cmp.

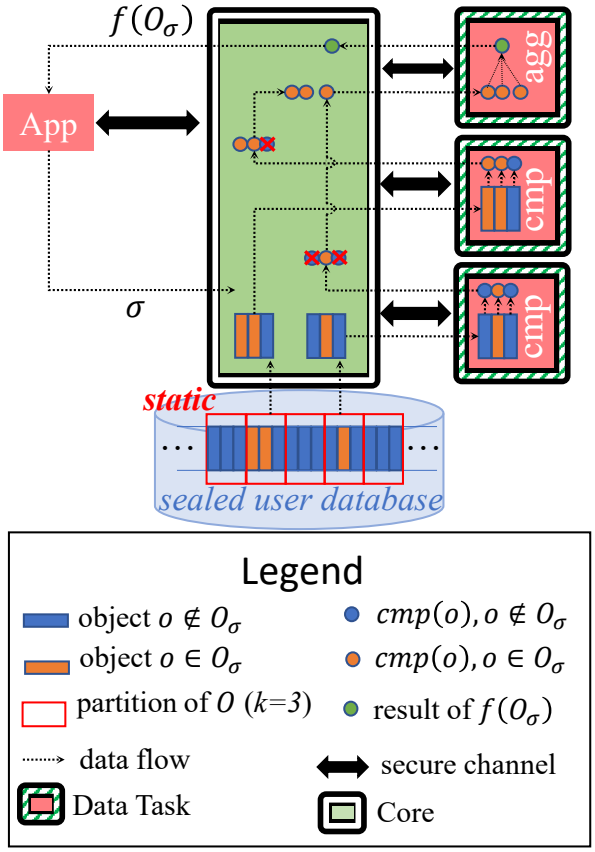


Figure 4.1: Decomposed execution with static partitioning.

Enforcement. To implement this Decomposed data tasks strategy, it is sufficient to add trusted code to the Core that implements an execution strategy consistent with this definition (Section 5.2 explains this in detail). Note that each partition being independent, all DT_i^{cmp} can be run in parallel depending on the parallelism capabilities of the system.

Impact on leakage. Any computation involves one or several partitions P_i of O. Due to our execution strategy leveraging stateless and deterministic Data tasks, evaluating cmp on a specific partition P_i is guaranteed to always produce the same result set $\{cmp(o) \mid o \in P_i\}$ and thus the same result $cmp(o)$ for any $o \in O$.

Hence, the Data set leakage for any partition P_i is bounded by $||cmp|| \cdot |P_i|$, regardless of the number of successive computations involving any $o \in P_i$. Consequently, the Data set leakage over a very large number n of computations on O is also bounded: ${L_f}^{n \to +\infty}(O) \leq \sum_i (||cmp|| \cdot |P_i|) = ||cmp|| \cdot |O|$.

Regarding Object leakage, for any P_i, the attacker has the liberty to choose which fragment of P_i to put inside each of the $|P_i|$ results produced by the Data Task DT_i^{cmp}. At the extreme, all $|P_i|$ fragments can concern a single object in P_i.

For any object $o \in P_i$, the Object leakage is thus bounded by ${L_f}^{n \to +\infty}(o \in O) \leq min(||cmp|| \cdot k, ||o||)$, with k the leakage factor equal to the maximum number of objects in any P_i.

'Green bonus' leakage analysis. As an example, we illustrate the impact on leakage with the Green Bonus scenario. Let us assume that $||cmp|| = 1$ (i.e., 1 bit to indicate if a GPS trace is a bike commute or not), $||agg|| = 6$ (i.e., 6 bits to count the number of monthly bike commutes with a maximum admitted value of 60) and $||o|| = 600$ (i.e., each GPS trace is encoded with 600 bits of information). Without any building block enforced on the execution of f, an attacker needs 100 queries to obtain an object o. By using a stateless Data Task for f, the number of queries to obtain o is (much) higher due to random leakage and o has to be contained in the input of each query. With a stateless and deterministic Data Task for f, the number of queries to obtain o is at least the same as with a stateless f but each query has to have a different input while still containing o. Finally, with a fully decomposed execution of f (i.e., $k = 1$) using stateless and deterministic Data Tasks, only $||cmp|| = 1$ bit of o can be leaked regardless of the number of queries.

4.4 . Conclusion

This chapter exposes our first proposals to tackle the leakage of data through the results of aggregate computations on our ES-PDMS platform. Presented in the form of security building blocks, they hinder the ability of Data Tasks to leak efficiently across computations by preventing them from relying on persistent state, randomness and the complete view of the input data.

These security building blocks greatly improve the capacity of our architecture to protect against leakage. From a potentially unlimited leakage of data, together they impose an upper bound on the amount of data that can be obtained by an attacker after an unlimited number of computations.

From the above formulas of this upper bound, a decomposed execution of $f = agg \circ cmp$ using stateless and deterministic Data Tasks is optimal in terms of limiting the potential data leakage, with both minimum data set and object leakages, when a maximum degree of decomposition is chosen, i.e., a partitioning at the object level by fixing $k = 1$ as leakage factor. However, reaching this minimal leakage requires to allocate at runtime one stateless and deterministic Data task per object $o \in O_\sigma$ involved in the computation, which has a negative impact on performance.

5 - Leakage Control - Efficient Evaluation Strategies

Contents

This chapter is devoted to the second part of our problem (see Section 3.3.4), addressing the main issue arising with a direct implementation of our security building blocks: performances. First, we describe two mechanisms aiming at improving the performances while maintaining the low data leakage offered by the building blocks. Then, we present three execution strategies leveraging both the building blocks and these mechanisms to run computations with low data leakage and acceptable performances.

5.1 . Performance Overhead

The building blocks presented in the previous chapter allow an evaluation of f with low and bounded leakage, minimal if $k = 1$. However, an evaluation strategy based on a direct implementation may lead to unrealistic performance (see Chapter 6) in the case of large sets of objects, mainly because (i) many unnecessary computations are needed (objects $o \notin O_\sigma$ must be processed, given the 'static' partitioning, if they belong to partitions containing objects $o \in O_\sigma$, see Figure 4.1), and (ii) too many data tasks must be allocated at execution (up to one per object $o \in O_\sigma$ to reach minimal leakage).

We need to overcome these obstacles and establish evaluation strategies with acceptable performance in practice, while still maintaining low data leakage. Therefore, we propose new execution strategies leveraging two mechanisms:

1. **Result reuse**, which avoids computations on objects that are not part of the input (objects $o \notin O_\sigma$) and opens the way to new strategies based on a 'dynamic' partitioning of the input objects of f;

   and

2. **Execution replay**, which allows applying coarser-grained partitioning schemes with a reduced set of Data tasks allocated during the execution, while keeping object leakage low to minimum.

5.2 . Decomposed Execution with Result Reuse

Result reuse consists in storing into the Core any new intermediate result $cmp(o)$ for any object o after its first computation and then reusing this value[1] for all subsequent evaluations of f taking o in input (i.e., $O_\sigma \ni o$).

Result reuse implies processing any raw database object o only once in its lifetime, without the attacker having the opportunity to consider again that object o as input of –potentially corrupted– function cmp, and thus without any means to further impact data leakage related to o. Hence, 'static' partitioning is not required anymore, and this allows adopting 'dynamic' partitioning schemes, where the set of –newly computed– objects part of the computation input O_σ can be partitioned and processed independently of other –already computed– objects in $O \setminus O_\sigma$, while still satisfying Definition 5.

This paves the way for a computation strategy based on (i) a *Generic* execution algorithm with Result reuse which is the common entry point for (ii) a dynamic computation strategy, namely the *Adaptive* execution algorithm presented here, the *Reverse-and-Replay* or the *Repartition-and-Replay* execution algorithms presented next. All these algorithms (the generic part and the computation strategy algorithms) are considered trusted and are part of the Core, while the codes of agg and cmp are considered untrusted and run therefore as Data tasks.

5.2.1 . Generic Execution with Result Reuse (Algorithm 1)

This code module run by the Core constitutes the generic entry point for any computation of f. It first determines the set of objects O_σ satisfying the computation (line 1) and splits it into two sets (lines 2-3): $O^+ \subseteq O_\sigma$ the objects that have already been computed in previous computations of f and $O^- \subseteq O_\sigma$ the objects selected for the first time. Then it constructs CMP_{O^+} by retrieving the cmp values stored for the objects in O^+ (line 4). For the objects in O^-, it triggers a *compute* process (line 6) based on the selected computation strategy (i.e., *Adaptive* presented below, *Reverse-and-Replay* introduced in Section 5.3.2

[1] In a PDMS context, we deal mainly with historical data (e.g., electricity traces, GPS histories, medical data, personal images, etc.). An implicit assumption considered in this thesis is that the personal database is managed in append only mode: objects are inserted or deleted, but not updated (see Section 5.4).

Algorithm 1 Generic execution with Result reuse (Core code)

Input: Querier a, session key K_a, predicate σ defining $O_\sigma \subseteq O$
Output: Value $v = agg\,(cmp(O_\sigma))$ result of computation

1: $O_\sigma \leftarrow \{o \in O \mid \sigma(o) = true\}$ ▷ objects in query scope
2: $O^+ \leftarrow \{o \in O_\sigma \mid cmp(o) \neq null\}$ ▷ objects with existing cmp values
3: $O^- \leftarrow \{o \in O_\sigma \mid cmp(o) = null\}$ ▷ objects with missing cmp values
4: $CMP_{O^+} \leftarrow \{cmp(o) \mid o \in O^+\}$ ▷ existing $cmp(o)$ values
5: **if** $O^- \neq \emptyset$ **then** ▷ compute missing cmp values
6: $CMP_{O^-} \leftarrow$ compute(O^-)
7: store CMP_{O^-} values in PDMS
8: **end if**
9: $DT^{agg} \leftarrow$ createDT(agg) ▷ create Data Task (agg code)
10: open(DT^{agg}) ▷ open secure channel (attestation)
11: send$(DT^{agg},$ sort $(CMP_{O^-} \bigcup CMP_{O^+}))$ ▷ send cmp values
12: $v \leftarrow$ receive(DT^{agg}) ▷ receive the result
13: killDT(DT^{agg}) ▷ kill DT^{agg}
14: **return** $encrypt(v, K_a)$ ▷ return encrypted result

or *Repartition-and-Replay* exposed in Section 5.3.3), and then stores the results for future use (line 7). Finally, a stateless and deterministic Data task DT^{agg} is instantiated to aggregate the entire set of all values $\{cmp(o) \mid o \in O_\sigma\}$ (lines 9-13) before sending the final result to the App encrypted with the shared session key K_a.

5.2.2 . Adaptive Execution (Algorithm 2)

This execution strategy implements the $compute$ function (see line 6 in Algorithm 1). It is called *Adaptive* as it leverages the results reuse to perform a 'dynamic' partitioning of O_σ, i.e., computed progressively based on each input of the queries in the workload, as opposed to the 'static' partitioning method in Section 4.3. Figure 5.1 illustrates the Adaptive strategy. Given an input O^- of database objects never computed before and a value k indicating a maximum cardinality, it builds a partitioning $\{P_i \mid i \in [1, m] \wedge |P_i| \leq k\}$ of O^- with $m = \left\lceil \frac{|O^-|}{k} \right\rceil$. Then, a set of m stateless and deterministic Data tasks is instantiated and each DT_i^{cmp} with $i \in [1, m]$ evaluates the function cmp on the partition P_i and produces a result set $\{cmp(o) \mid o \in P_i\}$. The final result CMP_{O^-} is the union of all the result sets.

5.2.3 . Leakage and Performance Analyses

Leakage analysis. The analysis detailed in Section 4.3 remains entirely valid for the Adaptive strategy. Now, instead of relying on static partitioning to impose

Algorithm 2 Adaptive execution (Core code)

Input: O^- a set of database objects, k the leakage factor
Output: CMP_{O^-} a set of $cmp(o)$ values for these objects

1: $m \leftarrow \left\lceil \frac{|O^-|}{k} \right\rceil$ ▷ number of partitions
2: $\{P_i \mid i \in [1, m]\} \leftarrow$ partitioning of O^- with $|P_i| \leq k$
3: $CMP_{O^-} \leftarrow \emptyset, CMP_* \leftarrow \emptyset$
4: **for** i in $[1, m]$ **do**
5: $DT_i^{cmp} \leftarrow$ createDT(cmp)
6: open(DT_i^{cmp})
7: send(DT_i^{cmp}, P_i) ▷ send partition
8: $CMP_* \leftarrow$ receive(DT_i^{cmp}) ▷ receive result set
9: killDT(DT_i^{cmp})
10: $CMP_{O^-} \leftarrow CMP_{O^-} \bigcup CMP_*$
11: **end for**
12: **return** CMP_{O^-}

determinism, the reuse of cmp results offers only a unique opportunity per object to leak information, thus the conclusions on Data set and Object leakages detailed in that section still hold. Note that the determinism of agg still requires sorting (line 11 of Algorithm 1).

Performance considerations. In terms of performance, Adaptive has two advantages compared to the Decomposed solution proposed in Def. 5: on the one hand, it allows to process only the database objects useful to the computation, i.e., objects $o \in O_\sigma$; on the other hand, the dynamic nature of Adaptive partitioning allows to reduce the number of involved Data tasks to $m = \left\lceil \frac{|O^-|}{k} \right\rceil$, with $O^- \subseteq O_\sigma$ the set of newly computed input objects. The same parallelism as Decomposed is possible with Adaptive, i.e., each iteration of the loop (lines 5 to 10) can be run in parallel.

However, to reach minimal leakage, it is necessary (as for the theoretical solution of Def. 5) to consider singleton partitions (i.e., a leakage factor $k = 1$), which imposes one Data task per newly computed object. Such a large number of Data tasks may proscribe this solution in practice, at least in scenarios (e.g., energy use-case) requiring various computations on large sets of objects, due to the performance overhead of creating enclaves to host the numerous Data tasks, as confirmed in our measurements in Chapter 6 (and in line with previous studies on data processing in TEEs like [197, 198]).

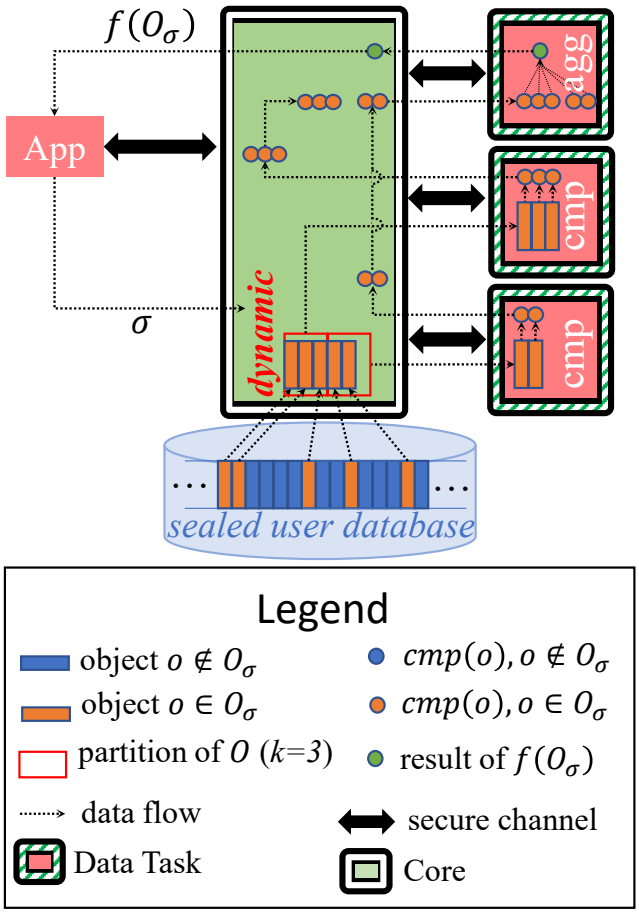


Figure 5.1: Evaluating f with Adapative Algorithm.

5.3 . Execution Replay

5.3.1 . Replay Introduction

Unlike previous strategies that impose the creation of a Data task for each object in the input to reach minimal leakage during the evaluation of cmp, the objective is to propose a solution leveraging execution replay in order to reduce the number of Data tasks involved in the execution while supporting low leakage factor. *Execution replay* mechanism consists in allocating a reduced number of stateless and deterministic Data tasks at runtime, themselves processing the input objects of O_σ several times, the minimality of the leakage being guaranteed by an equality check between the different results produced for the same object.

The general idea is as followed. Consider two sets of objects O_1 and O_2, with one single object o_j in common, i.e., $O_1 \cap O_2 = \{o_j\}$, and two deterministic Data tasks DT_1^{cmp} and DT_2^{cmp}, which respectively receive O_1 and O_2 as input and produce the results sets $CMP_1 = \{cmp(o) \mid o \in O_1\}$ and $CMP_2 = \{cmp(o) \mid$

$o \in O_2\}$ as output. If both Data tasks produced an identical result value for object o_j (i.e. $CMP_1[o_j] = CMP_2[o_j]$) then this result does not contain information about objects in $(O_1 \cup O_2) \setminus \{o_j\}$ (or this information is the same[2]).

In the following sections, we propose two strategies leveraging execution replay. The first one, *Reverse-and-replay*, computes the input set O^- object per object using the same two Data Tasks, in one order and in the reversed order. The second one, *Repartition-and-replay* replays the execution multiple times with different partitioning. Each strategy is efficient in its own physical implementation of our architecture: Reverse-and-replay is more suited when the Core and Data Tasks are executed on the same machine i.e., communication latency is very low, whereas Repartition-and-replay outperformed it when remote Data Tasks are used.

5.3.2 . Reverse-And-Replay Strategy (Algorithm 3)

Given an input set of objects $O^- \subseteq O_\sigma$, the Reverse-and-replay execution works as follows (see Algorithm 3 and an overview in Figure 5.2). The Core instantiates two stateless and deterministic data tasks DT_1^{cmp} and DT_2^{cmp} and uses these data tasks in pipeline to compute $cmp(o)$, for all $o \in O^-$, one of the pipeline being reversed from the other (lines 7 to 14). More precisely, the Core provides the input set to DT_1^{cmp} in the sequence order (input: $o_1 \rightarrow o_2 \rightarrow ... \rightarrow o_n$) one object at a time, DT_1^{cmp} producing a result after each input (output: $res_1 = cmp(o_1) \rightarrow res_2 = cmp(o_2) \rightarrow ... \rightarrow res_n = cmp(o_n)$). DT_2^{cmp} processes exactly the same input but in the reverse order (input: $o_n \rightarrow o_{n-1} \rightarrow ... \rightarrow o_1$; output: $res'_n \rightarrow res'_{n-1} \rightarrow ... \rightarrow res'_1$). Finally, the Core checks for results equality, i.e. $\forall j \in [1, n]$, $res_j = res'_j$ (line 17) and, in the positive case, returns the set of computed results.

Leakage analysis. Given the pipelined execution, the results res_j and res'_j produced by the Data tasks for any object o_j can only contain information from objects previously processed by the Data task, i.e., about $\{o_x \mid x \in [1, j]\}$ for res_j and about $\{o_x \mid x \in [j, n]\}$ for res'_j. When these results are identical (i.e., $\forall j$, $res_j = res'_j$), they do not contain information about $\{o_x \mid x \in [1, j[\ \}$ nor about $\{o_x \mid x \in]j, n]\}$, hence $res_j = res'_j$ only discloses information about o_j. Thus, the Reverse-and-replay execution offers a minimal *object leakage*. Combined with Result reuse benefits, it guarantees a minimal *data set leakage* too, due to the uniqueness of $cmp(o_j)$ regardless of the number of computations including o_j.

Performance considerations. The main advantage of this strategy is that it requires only two Data Tasks regardless of the number of selected objects for computation. This is much fewer than with the Adaptive strategy, which requires

[2]We assume that objects do not have a shared part. Note also that we focus on cases where the attacker's goal is to reconstruct raw database objects, where each leaked data fragment would be associated with some metadata, e.g., indicating an identifier for the object containing the fragment or fragments' position within the object. Hence, even if two distinct data fragments contain the same information, the associated metadata are different leading to different codification of the leakages.

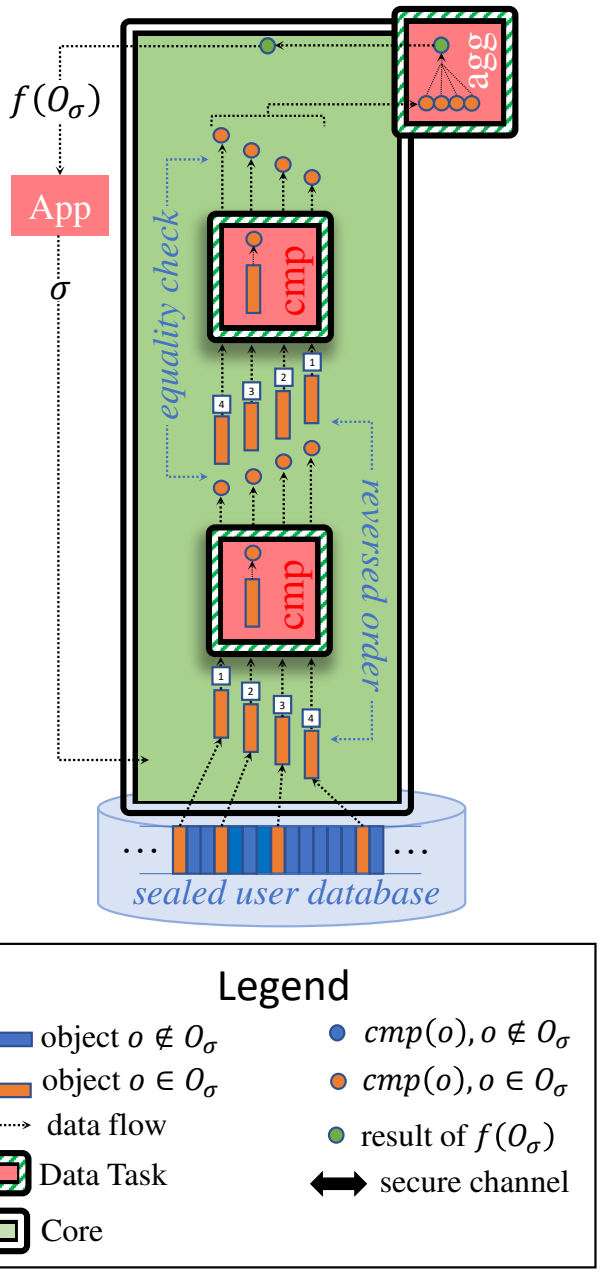


Figure 5.2: Evaluating f with Reverse-and-Replay Strategy.

Algorithm 3 Reverse-and-replay execution (Core code)

Input: $O^- = \{o_j \mid j \in [1,n]\}$ a set of n database objects
Output: CMP_{O^-} a set of $cmp(o)$ values for these objects

1: $CMP_1 \leftarrow \emptyset, CMP_2 \leftarrow \emptyset$
2: $DT_1^{cmp} \leftarrow$ createDT(cmp)
3: open(DT_1^{cmp})
4: $DT_2^{cmp} \leftarrow$ createDT(cmp)
5: open(DT_2^{cmp})
6: **for** j in $[1,n]$ **do**
7: send(DT_1^{cmp}, o_j) ▷ send jth object to 1st DT
8: $res_j \leftarrow$ receive(DT_1^{cmp})
9: $CMP_1 \leftarrow CMP_1 \bigcup \{res_j\}$ ▷ add result to 1st resultset
10: send(DT_2^{cmp}, o_{n-j+1}) ▷ send $n-j$th object to 2nd DT
11: $res'_{n-j+1} \leftarrow$ receive(DT_2^{cmp})
12: $CMP_2 \leftarrow \{res'_{n-j+1}\} \bigcup CMP_2$ ▷ add result to 2nd resultset
13: **end for**
14: killDT(DT_1^{cmp})
15: killDT(DT_2^{cmp})
16: **if** $CMP_1 \neq CMP_2$ **then** ▷ if result sets are not equal
17: **return** ERROR
18: **end if**
19: **return** CMP_1

one Data Task per –newly computed– input object for a minimal leakage. While Reverse-and-replay induces an additional cost for computations because every object has to be computed twice, it is not critical compared to the benefits in leakage protection (especially since pipelined Data Tasks can easily be executed in parallel, see Chapter 6).

However, this strategy does not perform well in all contexts. As stated in Section 3.1.1, one possible physical implementation of our ES-PDMS architecture is to rely on remote Data Tasks, i.e., running on another machine than the one hosting the Core, the two being connected via the Internet. In this scenario, communication latency is two or three order of magnitude slower than inter-process communication on the same machine. The pipeline behaviour of the Reverse-and-Replay strategy leads to many communications between the Core and the Data Tasks, which heavily suffers from network latency as detailed in Chapter 6.

5.3.3 . Repartition-And-Replay Strategy (Algorithm 4)

Given the performance overhead of the Reverse-and-Replay strategy when dealing with remote Data Tasks, it is important to devise another strategy addressing

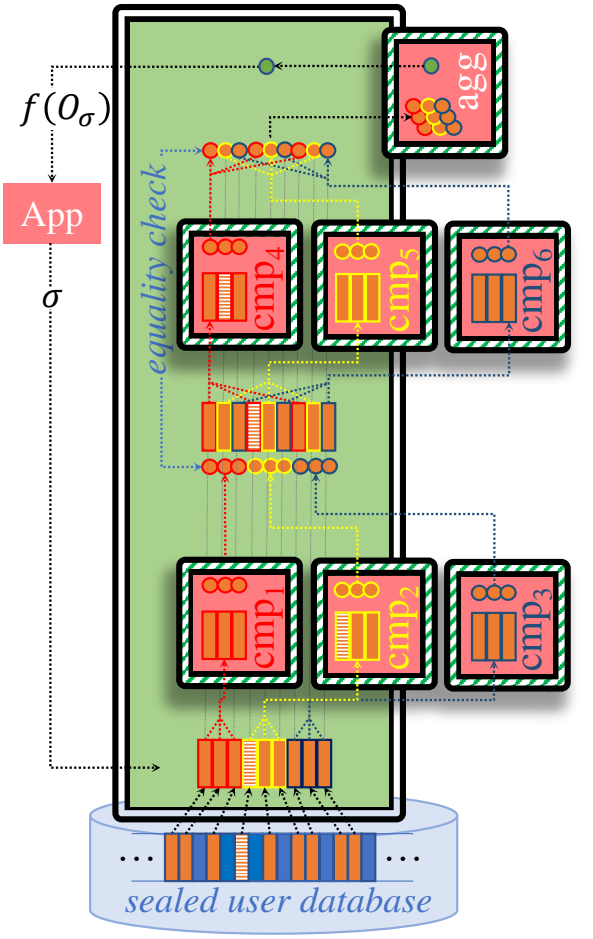
this weakness. Like Reverse-and-Replay, this strategy must have better performances than Adaptive when dealing with a low leakage factor, but it must not be severely degraded by communication latency.

The weakness of Reverse-and-Replay over latency is due to the fact that each input object causes two communications between the Core and the Data Task: one to send the object for computation and one to receive its associated result. Reducing the number of communications means that objects have to be sent (and computed) by batch, comparable to the Adaptive strategy with $k > 1$, while still being able to efficiently compute with a low leakage factor.

Based on this idea we designed the following strategy called Repartition-and-Replay. To guarantee by construction an evaluation with a leakage bounded by a leakage factor k (as in Adaptive), the idea is to make a partitioning of O^- into m disjoint partitions $\{P_1, P_2, ..., P_m\}$ of (approximately) equal (and large) size, and use a set of Data tasks $\{DT_i^{cmp} \mid i \in [1, m]\}$, each one producing a set of results $\{cmp(o) \mid o \in P_i\}$ for one P_i. This first step is similar to the Adaptive strategy. However unlike Adaptive, this step is replayed several times, each time with a different m-partitioning of O^-. The partitioning is designed such that after R replays, for any object $o \in O^-$, there remains at most $k = \left\lceil \frac{|O^-|}{m^R} \right\rceil$ common objects in the intersection of the R successive partitions containing o. The value of k corresponds to the leakage factor indicating the number of results in which malicious code may inject information about given object o. Therefore, a number of replays $R = \lceil log_m(O^-) \rceil$ guarantees a minimum $k = 1$ value, where any object $o \in O^-$ is the only common object in the intersections of the R partitions containing o. The corresponding algorithm is illustrated in Figure 5.3 with $k = 1$, $m = 3$ and a set O^- of nine objects. Note for example that the hatched object o_j is first included in a yellow partition processed by Data task cmp_2, then in a red partition processed by cmp_4, and is thus the only object at the intersection of these (yellow and red) parts, such that if the result produced for this object during the computation of each partition is identical, it depends only on this object. The algorithm works as follows:

Repartition-and-Replay execution. In Algorithm 4, given an input objects set O^-, a leakage factor k and a number of partitions m, the number of replay iterations R is computed to reach the expected leakage factor (line 1). Then at each iteration $r \in [1, R]$, the input objects set O^- is partitioned into m partitions such that the object having the index j in O^- is included within partition P_i if and only if $(\lfloor j \cdot m^r / n \rfloor \mod m) = i$ (line 4). A stateless and deterministic Data task DT_i^{cmp} is created for each partition and computes cmp on $\{o_j \mid o_j \in P_i\}$. The Core checks that the results $res^*_{o_j}$ obtained for each $o_j \in O^-$ is consistent with the previously computed values res_{o_j} (line 9). Finally, the complete set of results is returned (line 14).

Leakage analysis. Repartition-and-Replay execution guarantees by construction that after a number of replays $R = \lceil log_m(n/k) \rceil$ (with $n = |O^-|$), any object



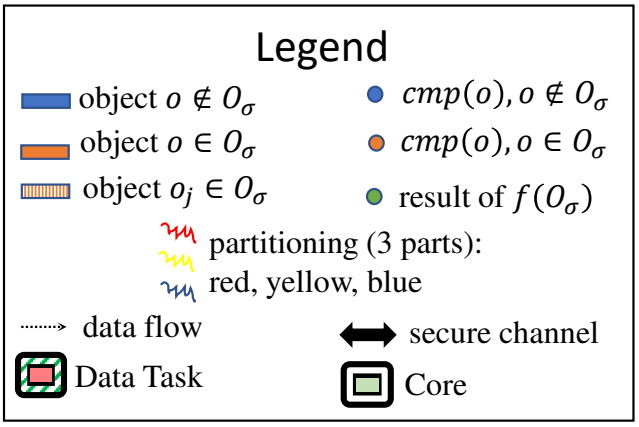


Figure 5.3: Evaluating f with Repartition-and-Replay Algorithm.

Algorithm 4 Repartition-and-Replay execution (Core code)

Input: $O^- = \{o_j \mid j \in [1, n]\}$ a set of n database objects, k the leakage factor, m the number of partitions per replay
Output: CMP_O^- a set of $cmp(o)$ values for these objects

1: $R \leftarrow \lceil log_m(n/k) \rceil$ ▷ number of replays
2: **for** r in $[1, R]$ **do** ▷ for each replay iteration
3: **for** i in [1, m] **do** ▷ for each partition
4: $P_i \leftarrow \{o_j \in O^- \mid (\lfloor j \cdot m^r / n \rfloor \mod m) = i\}$
5: $DT_i^{cmp} \leftarrow$ createDT(cmp)
6: send(DT_i^{cmp}, P_i)
7: $\left\{res^*_{o_j} \mid o_j \in P_i\right\} \leftarrow$ receive(DT_i^{cmp})
8: killDT(DT_i^{cmp})
9: **if** $\exists o_j \in P_i \mid res^*_{o_j} \neq res_{o_j}$ (res_{o_j} from previous replays) **then**
10: **return** ERROR
11: **end if**
12: **end for**
13: **end for**
14: **return** $CMP_{O^-} = \left\{res_{o_j} \mid j \in [1, n]\right\}$

$o \in O^-$ has been processed R times as part of R different partitions. Moreover, the intersection of all partitions containing o is indeed equal to a set of objects of cardinality (at most) k. Since all the successive results associated with o for each of these partitions was checked to be identical, information about o can only leak into k results. Fixing $k = 1$ guarantees a minimum Object leakage (Def. 2) as in Adaptive. And as Adaptive, the Data set leakage (Def. 1) is minimum due to Result reuse, ensuring the uniqueness of $cmp(o)$ for any o regardless of the number of computations including o.

Performance considerations. Compared with Adaptive, Repartition-and-Replay involves an increased number of computations: for each object o, $cmp(o)$ is evaluated R times instead of once with R being at most $\lceil log_m(n) \rceil$. However the cost of these re-computations is widely compensated by the reduction in the number of Data Tasks as seen in Chapter 6: from at most n Data Tasks with Adaptive, Repartition-and-Replay involves at most $m \cdot \lceil log_m(n) \rceil$ Data Tasks. By analysing the derivative (See Appendix A), we can see that this number is minimal when $m = 3$. Thus, in an example of a computation of 30.000 objects (comparable to one of the datasets presented in Section 6.1.2), only 30 Data Tasks are required to achieve a leakage factor of $k = 1$ with Repartition-and-Replay.

This strategy was introduced to overcome the limitations of Reverse-and-Replay in the case of remote Data Tasks. Although it uses $m \cdot log_m(n)$ Data Tasks

instead of two, Repartition-and-Replay highly reduces the number of communications involved between the Core and the Data Tasks, from $4 \cdot n$ to $2 \cdot m \cdot log_m(n)$, with m small, because inputs are processed by batch. This has a significant impact on performances compared to Reverse-and-Replay when communication latency is involved as depicted by Section 6.2.4.

5.4 . Conclusion

In this chapter, we presented solutions to tackle the main shortcoming of the security building blocks defined in Chapter 4, namely performances. First, we proposed two mechanisms to complete these building blocks by dealing with important slowdown factors:

- **Result Reuse** which avoids computing objects that are not part of the current input.
- **Execution Replay** which reduces the number of Data Tasks involved in a computation to enable the use of low leakage factors with reasonable performances.

Then, we introduced three execution strategies which are execution plans for the considered computations $agg\left(cmp\left(O_\sigma\right)\right)$ implementing the building blocks and these mechanisms.

- The first one is the Adaptive strategy which leverages Result Reuse and processes the object by batch. Its advantages are that it only computes each object once and the leakage factor can be configured at will, this strategy benefiting the most from large leakage factors. However, small value for the leakage factor severely degrades its performances.
- The second one is the Reverse-and-Replay strategy which uses both Result Reuse and Execution Replay, processing the input object per object, twice. It only employs two Data Tasks to compute the input and provides the best leakage factor. Yet, the resulting number of communications entails a strong dependency between its performances and the communication latency, which is an important factor when exploiting remote Data Tasks.
- The third strategy is Repartition-and-Replay which also uses both Result Reuse and Execution Replay but processes the input several times by batch. As its main benefits, the leakage factor can be configured at will and the number of communications is low: a logarithm of the number of objects. Its shows the best performances over the three strategies when a low leakage factor and significant latency are involved.

Chapter 6 describes in detail the performance analysis of these strategies.

To the best of our knowledge, our work is the first to offer solutions to tackle the leakage of data through computation results on an ES-PDMS. As such, complementary solutions, challenges and research opportunities can be derived from this thesis. A first limitation is that our study considers a single function $f = app \circ cmp$ for a given App. For Apps requiring several computations and thus several codes for cmp and agg, our leakage analysis still applies for each cmp, but the total leak that can be obtained by an attacker controlling the App is the cumulation across the set of functions. It is also the case if the Apps collude. Also, this study does not discuss data updates. Personal historical data (mails, photos, energy consumption, trips) is append-only (with deletes) and is rarely modified. From the viewpoint of the proposed strategies, an object update can be seen as the deletion and reinsertion of the modified object. And at each reinsertion, the object is exposed to some leakage. Hence, with frequently updated and queried objects, new strategies may be envisioned.

More important research challenges have been envisioned and described in details in Section 8.2.

6 - Validation

Contents

This chapter presents our extensive performance evaluation of the security building blocks and execution strategies introduced in Chapter 4 and Chapter 5, respectively. Our evaluation studies the efficiency and scalability of these proposals based on an implementation using Intel SGX, two real data sets and representative computations.

6.1 . Experiment Materials

6.1.1 . Experimental Platform

For all experiments, we use a server with an Intel Xeon E-2276G 6-cores @3.8GHz supporting SGX 1-FLC. Out of the 64 GB of RAM available on the server, 128 MB are reserved for Intel SGX with 93.5MB usable by enclaves. It runs Ubuntu 18.04 (kernel 4.15.0-142) with the SGX DCAP driver v1.41. This portrays the scenario where the PDMS would be deployed entirely (i.e. Core and Data tasks) on the Cloud. Then, to grasp the context of a PDMS running (entirely) on a user device, we also measured performances on a personal computer having an Intel Core i5-9400H @2.5GHz 4-Cores also supporting SGX 1-FLC with 94MB usable by enclaves. The performances on this PC were similar to the ones on the server, with an additional computational overtime of approximately 10% due to a slower CPU, thus they are not shown here. Finally, to capture the case where the Core runs on another machine than the Data Tasks, the two being connected by

	Energy	GPS
Number of data points	2 075 259	24 876 978
Number of objects	34 587	18 670
Object size (data points)	60	1 332
Object size (bytes)	720	31 968
Task for cmp	Integral	Length
Result size of cmp (bytes)	4	4
Task for agg	Average	Sum
Result size of agg (bytes)	4	4

Table 6.1: Considered Data and Query sets

standard internet connection, we simulated the addition of communication latency between them. Software-wise, we used our own implementation of a PDMS using Intel SGX. The details of this implementation are given in Chapter 7.

6.1.2 . Data Sets

We use two public data sets of personal data (see Table 6.1). The first [199] contains the electric power consumption of a household minute by minute over a period of four years. Each object is a time series containing the consumption of one hour (i.e., 60 data points). The second [200] data set contains more than 18.000 GPS trajectories from 182 users using different transportation modes (e.g., car, bus, taxi, bike, walk) over more than five years. For scalability reasons, we consider the complete set as if it was generated by a single user. Each trajectory has different spatial and temporal length with 1332 GPS points for one trajectory on average. The Core stores each trajectory as a an object, making each GPS object 44 times larger on average than the electricity objects. We name these data sets Energy and GPS in the subsequent sections. For both data sets, each object is associated with the corresponding date interval used by the Core to select the objects required for a computation.

6.1.3 . Query Sets

We give the App the right to request the execution of two UDFs, one for each data set. For the Energy data set, the App is able to receive the average of the energy consumption of the user for any time interval(s) provided by the App at execution time through σ. This corresponds to the Energy example described in Chapter 1. For the GPS data set, the App can request the sum of the length of GPS trajectories, also for any time interval(s). To carry those computations, appropriate Data tasks are used: two codes for Data tasks implementing a cmp function, one computing the integral of electric consumption time series and the other computing the length of GPS trajectories ; and two codes for agg Data tasks computing either an average or a sum of integers. All Data tasks produce as output an integer of four bytes to preserve precision. Smaller result sizes can be

considered depending on the required accuracy of the result, but this would have a negligible impact on performance and is therefore not considered in this chapter. The same cmp and agg functions have also been integrated into the Core to enable a comparison without Data Tasks detailed in Section 6.2.5.

6.1.4 . Experimental Approach

Our experimental approach consists in three steps. First, we measure the computation times of $agg \circ cmp$ using the Decomposed execution (see Section 4.3) with a maximum degree of decomposition (i.e., a leakage factor of $k = 1$) over different selectivity (i.e., different σ) and deduce the main bottlenecks. Then, we evaluate and compare the trade-off between leakage and performances offered by Adaptive, Reverse-and-Replay and Repartition-and-Replay on both data sets, different leakage factors and communication costs. Finally, we consider the performance of these strategies over a query workload.

In the rest of this chapter, the computation time includes the different steps of the execution:

1. the processing of the SQL query by the Core, selecting the necessary objects
2. the creation of the enclaves for all Data Tasks
3. the attestation of each Data Task by the Core
4. the sending of the objects to cmp Data Task(s), including encryption and decryption
5. the computation of cmp by the Data Task(s)
6. the sending of the results to the Core, including encryption and decryption
7. the sending of these results to one agg Data Task, including encryption and decryption
8. the computation of agg by this Data Task
9. the sending of the final result to the Core, including encryption and decryption

The duration of communications between the App and the Core are not included as not relevant to evaluate the performances of our strategies. Communications between the Core and the Data Tasks take place on the same machine, except for Section 6.2.4. The extra cost caused by the paging mechanism of Intel SGX when the RAM is full (see [93]) has not been included in our measurements. To do so, we inferred the times for computations requiring more than 94MB of RAM, the limit of RAM usable by enclaves on our CPU model, as this limit is meant to be largely extended and have already been set to 512GB per CPU socket in the last generation of Intel XEON Processors [118]. All reported computation times are the average of ten computations querying the same number of objects.

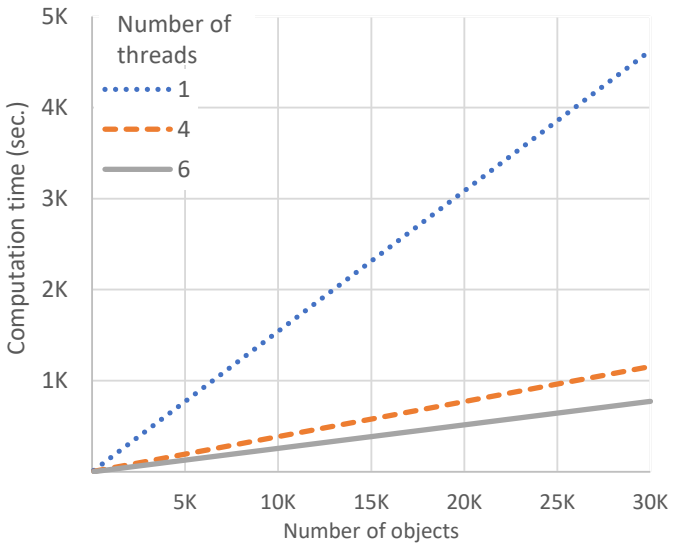


Figure 6.1: Decomposed k=1 – Energy

6.2 . Computation Scalability and Leakage

In this study, we are interested in the trade-off between leakage prevention and performances of the Adaptive and the two Replay strategies. Particularly, we want to assess the capability of these strategies to reach the minimum leakage factor ($k = 1$) for a computation with limited impact on its performance. As we are not aware of any existing solution for the studied problem to compare it with, we consider as baseline the Decomposed execution with minimum leakage factor and measure its performance first.

6.2.1 . Decomposed with Minimum Leakage

Figure 6.1 exposes the computation time with Decomposed execution set up with the minimal leakage $k = 1$ on an increasing numbers of objects from the Energy data set. As expected, its computation time is very high even with a relatively low number of objects (e.g., it exceeds 1 sec. with 6 objects processed) and it increases linearly. Benefiting from the multi-core platform, we also consider executions with an optimal number of Data tasks launched in parallel (processed using 6 threads, the number of CPU cores of the machine). Note that even with more CPU cores available, in practice the number of simultaneously running enclaves is limited with SGX depending on enclaves' memory footprint [201, p.6]. Despite all our optimization efforts, the computation time remains too high. We obtained similar results with the GPS data set. Unsurprisingly, Decomposed execution is impractical when configured with the minimum leakage factor.

Costs breakdown. Figure 6.2 details the main execution costs of the Decomposed execution (with $k = 1$) for a computation of 5000 objects. As expected,

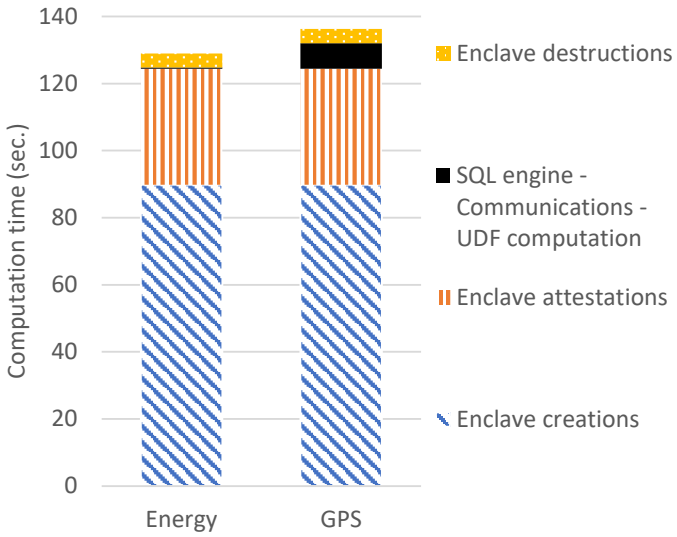


Figure 6.2: Decomposed k=1 breakdown – Energy

Data tasks creation and attestation are the main costs, representing more than 91% of the execution time. Indeed, on Intel SGX the creation of enclaves incurs a fixed cost depending notably on the number of code pages of the enclave [93]. This cost is approximately 110ms in our tests. Moreover, the Core has to establish a secure TLS channel with each Data task to authenticate it and securely send input data and retrieve results. This takes approximately 40ms per enclave in our tests. These costs are multiplied by the number of Data tasks (one per object in this strategy configuration) hence the high overhead. In comparison, the cost induced by the Core SQL Engine (to compute O_σ), communications and computation of the user-defined functions (all executions of agg and cmp functions) are marginal. We also note that these data related costs are 23 times higher with the GPS data set compared to Energy, because the GPS objects are larger and thus incur higher data transfer and computation times. Given the cost decomposition, minimizing the number of Data tasks involved in the computation is paramount.

6.2.2 . Strategies with Fixed Input Size

To assess the capability of our execution strategies (Chapter 5) to efficiently handle computations, we first compare their performances for the same number of input objects with a variable leakage factor. In all the following sections of this chapter, the execution strategies benefit from an optimal number of Data Tasks launched in parallel, i.e. 6 on our machine.

Figure 6.3 and 6.4 compare the performances of Adaptive, Reverse-and-Replay and Repartition-and-Replay depending on the value of k for the Energy dataset.

Note on Reverse-and-Replay. While the original algorithm for Reverse-and-Replay (see section 5.3.2) involves exactly two Data Tasks and processes the input

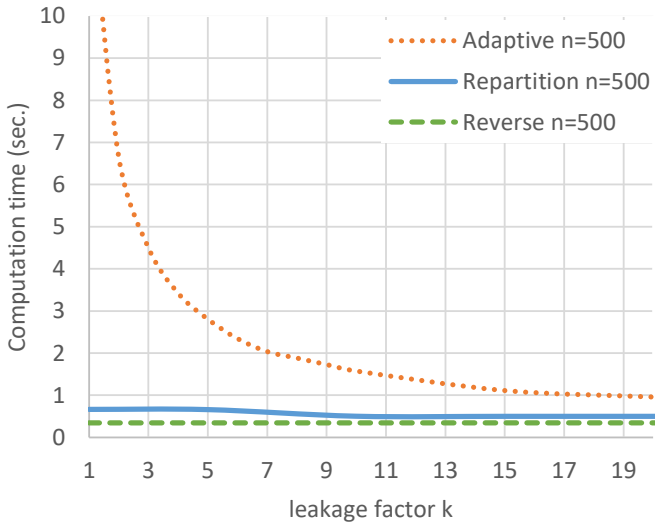


Figure 6.3: Comparison on 500 objects – Energy

object per object, we slightly modified it in our performance evaluation to benefit from the parallelism and to support leakage factors k different than 1. First, instead of relying on only two Data Tasks, we rely on 6 to match the number of CPU cores of the machine. In a computation on objects noted $\{o_j \mid j \in [1, n]\}$, the first Data Task will compute objects $\left\{o_j \mid j \in \left[1, \frac{n}{3}\right]\right\}$ in this order. The second and the third Data Task will compute respectively $\left\{o_j \mid j \in \left[\frac{n}{3} + 1, \frac{2n}{3}\right]\right\}$ and $\left\{o_j \mid j \in \left[\frac{2n}{3} + 1, n\right]\right\}$. The other three Data Tasks will compute the same sets but in the reversed order. On the other hand, instead of computing these sets object per object, we allow the strategy to process objects k by k, thus imposing a leakage factor of k.

Using *k=1*. These figures attest to the good performance of both replay strategies compared to Adaptive: For 500 objects, Repartition-and-Replay spends 660ms to handle a leakage factor of $k = 1$ while Reverse-and-Replay spends 343ms, thus being respectively 20 and 38 times faster than Adaptive taking 13.1s. For 5000 objects, it is 1.2s for Repartition-and-Replay, 663ms for Reverse-and-Replay and 128s for Adaptive, thus showing that adding one order of magnitude of objects only doubles the cost of both Replay strategies while it increase the cost of Adaptive by one order of magnitude. Both charts include the time spent to retrieve the objects from the database, namely 30ms for 500 Energy objects and 305ms for 5000. These performances are due to the fact that, in this case of $k = 1$, the number of data tasks is much lower with both Replay strategies: Repartition-and-Replay uses a number of Data Tasks that is proportional to a logarithm of the number of objects; Reverse-and-Replay employs two (or six, see the note above) Data Tasks; while Adaptive involves a number of Data Tasks linearly proportional to the number of objects.

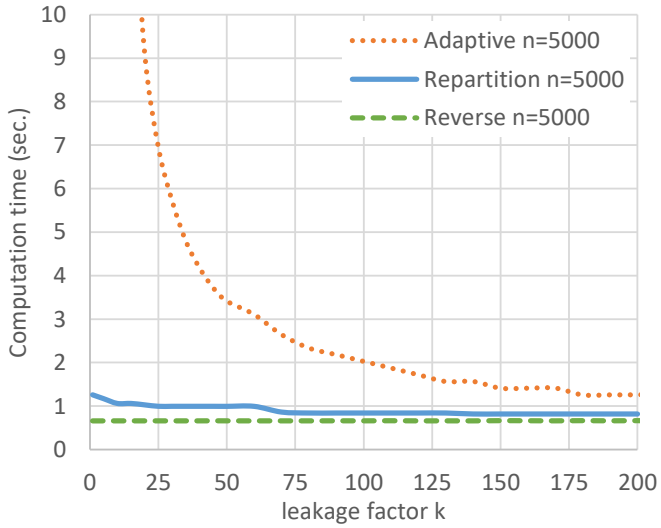


Figure 6.4: Comparison on 5000 objects – Energy

Increasing *k*. Increasing the leakage factor worsens the Object leakage (increasing the leakage factor of one order of magnitude increases the Object leakage of one order of magnitude, see Chapter 4). However, not all kinds of computation require a leakage factor of 1, thus we study the benefits in performance that can be obtained by using larger leakage factors. Increasing the leakage factor k leads to better performance for Adaptive and Repartition-and-Replay because fewer Data tasks are required. The improvement is invisible for Reverse-and-Replay since the same number of Data Tasks is involved, only the number of communications between the Core and Data Tasks is reduced which is a negligible cost when dealing with small objects and co-located Core and Data Tasks on the same machine.

GPS data set. Figure 6.5 depicts the strategies comparison on the GPS data set for 5000 objects. Because the GPS objects are bigger in size (44 times on average), the extra costs of selecting, transmitting and processing objects has a direct impact on the computation time: At $k = 1$, Repartition-and-Replay spends 11s to process the computation, 8 times more than with the Energy data set, the longer processing time being amplified by the fact that this strategy computes objects multiple times. When the leakage factor is high, e.g. $k \geq 100$, the performance of Adaptive becomes very similar to Repartition-and-Replay because in this case their behaviors matches: Repartition-and-Replay runs only one partitioning, without doing any replay, which is equivalent to Adaptive. For the same number of objects, small leakage factors can be handled quicklier with Repartition-and-Replay or Reverse-and-Replay than with Adaptive.

6.2.3 . Strategies with Fixed Performance

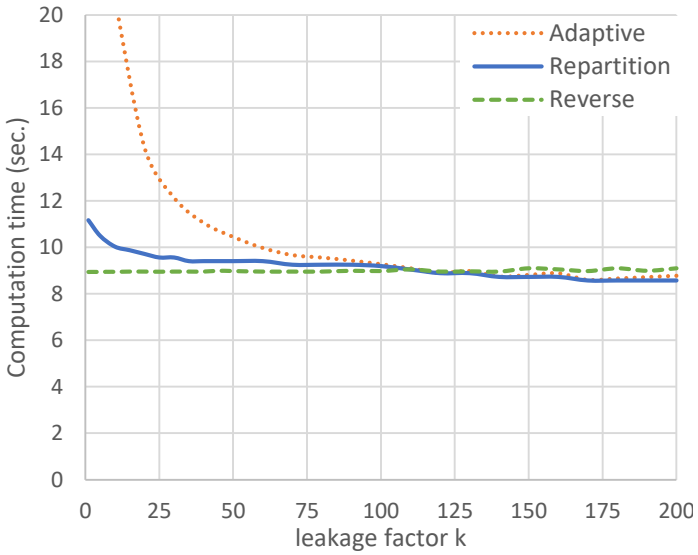


Figure 6.5: Comparison on 5000 objects – GPS

Figure 6.6 exposes the leakage concentration to be expected on a computation if no more than one second can be spared. Beyond 600 objects, the Adaptive strategy cannot handle a k value lower than 20 under one second. Repartition-and-Replay and Reverse-and-Replay strategies handle respectively at most 4700 and 9700 objects with a k value lower than 20 under one second. Within a five second limit (the graph is omitted), the delivered k value from the Adaptive strategy increases linearly, reaching 20 with only 3400 objects. In opposition, both Replay strategies can handle all the 30000 objects of the data set with the minimal leakage factor under five seconds.

For the GPS data set (see Figure 6.7), the time constraint is more limiting: Adaptive cannot handle more than 1600 objects with a k value lower than 20 under five seconds while Repartition-and-Replay and Reverse-and-Replay can handle up to 2400 and 2700 respectively. Under a time-limit constraint, both Replay strategies are thus able to compute objects with a smaller leakage factor than Adaptive.

6.2.4 . Latency Impact

Given our experimental platform, the Core and the Data Tasks are executed on the same machine. However, as explained in Section 3.1, this is but one possible physical implementation of our PDMS architecture. Another possible implementation leverages remote Data Tasks, hosted on Cloud servers, to carry out the computations. This leads to higher communication costs and calls for a separate analysis.

Figure 6.8 compares the strategies on 5000 objects of the Energy data set in a distributed setting, i.e., for non-colocated enlaves running the Core and the Data tasks where the Core runs on a PDMS user own device at home and the Data

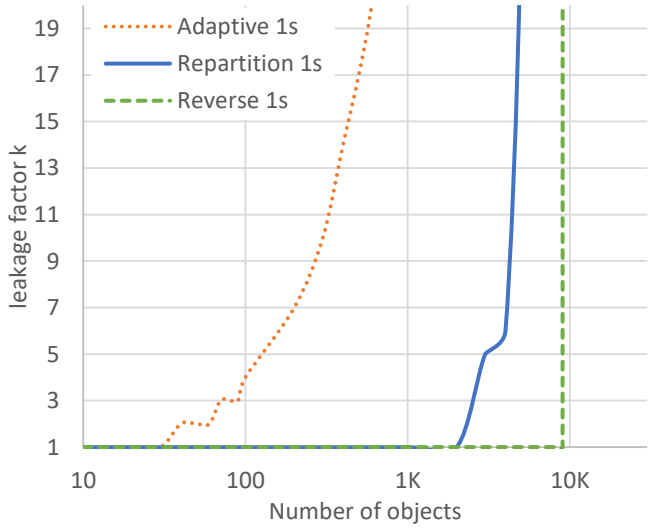


Figure 6.6: Comparison on fixed time – Energy

tasks are spawned when needed on a cloud server. We simulate a network link with a latency of 33ms and a bandwidth of 60Mbps representing average values for domestic Internet in France [202]. The results show that this setting affects differently the three strategies. As expected, Reverse-and-Replay is proportionally the most affected because of its pipelined data transmission between the Core and Data tasks, thus suffering from the latency for each computed object at $k = 1$. The other two strategies are less impacted thanks to their batch behavior. In this setting, Repartition-and-Replay is 55 times more efficient than Reverse-and-Replay, spending $2s$ on the computation compared to 110s. With these parameters, Repartition-and-Replay is faster than Reverse-and-Replay when the latency is greater than 270μs. Increasing k leads to fewer communications, thus significantly improving the computation time of all strategies. By efficiently providing support for low leakage factors and being the least impacted by latency, the Repartition-and-Replay method is therefore the preferred method in a distributed settings or generally when latency is involved.

6.2.5 . Execution Costs with Query Workloads

As expressed by our examples of scenarios (see Chapter 1), the same App can be interested to request several computations on different (but not necessarily disjoints) subsets of queryable objects. Thus it is important to study the performances of our strategies over a workload of queries. Figure 6.9 shows the evolution of the computation time over a workload of 250 queries on the GPS data set, each strategy being configured with a minimum leakage factor ($k = 1$). The workload is generated so that each computation processes objects belonging to 10 randomly selected time intervals, ranging in size from 2 to 24 hours. In this graph, the data

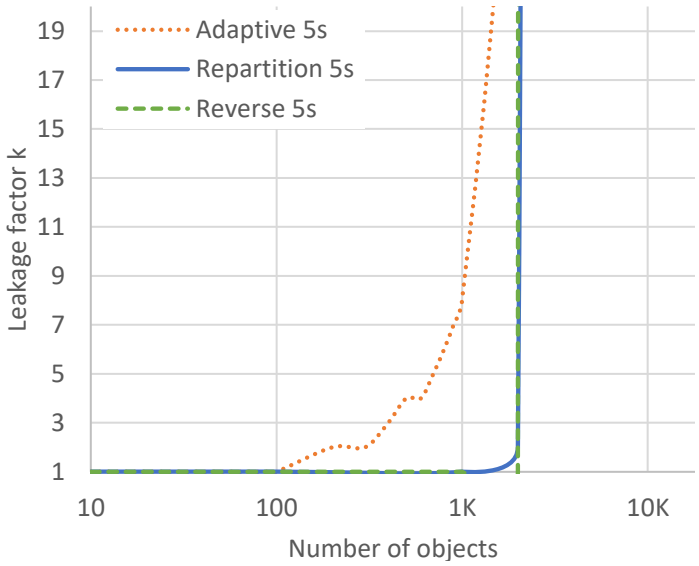


Figure 6.7: Comparison on fixed time – GPS

for 'Core only' correspond to an execution of the same function $f = agg \circ cmp$ within the Core of the PDMS itself, thus without the costs of creating, attesting and destroying Data Tasks, nor the encryption and sending of objects and results. This represents the case where the security of the UDF code would have been fully verified, and could thus be added to the PDMS Trusted Computing Base (TCB). While this may not be a realistic method for ensuring extensiveness nor compatible with the considered trust model described in Section 3.2.2), it allows establishing a lower bound in terms of performance and evaluating the overhead of using Data tasks.

From the first tenths to the hundredth query, Reverse-and-Replay is at most one order of magnitude more efficient than Adaptive, with Repartition-and-Replay being close to Reverse-and-Replay's performances (in average 47% higher). Then, the Result Reuse mechanism (See Section 5.2) benefits to all execution strategies, with fewer objects to be computed as queries are processed. The gap between both Replay strategies and Adaptive becomes insignificant after 175 queries. On another hand, Reverse-and-Replay is at most 85% more expensive than 'Core only', thus signifying that the cost of using Data Tasks, ensuring the extensiveness of the architecture, with the lowest leakage factor is less than a 2x overhead. On a query workload, both Replay strategies are thus the best candidates to ensure minimum leakage while being close to the ideal 'Core only' performance.

6.3 . Conclusion

The experimental evaluation presented in this chapter was aimed at testing the quality of our proposals on a PDMS implementation with real world data sets.

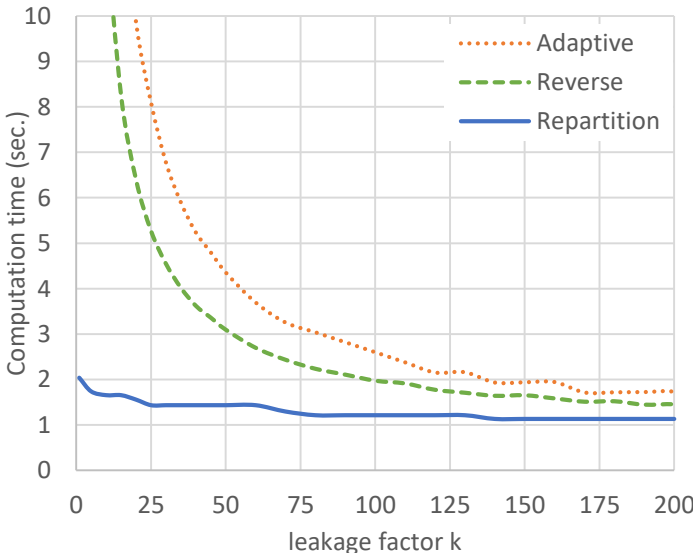


Figure 6.8: 5000 objects with bandwith and latency – Energy

Given that we address a data leakage problem, it was important to verify the ability of our proposals to tackle leakage while being efficient in terms of performances.

First, we attested that the direct implementation of our security building blocks (Chapter 4) was impractical in performances when configured with a maximal leakage protection. The main slowdown factor is the number of Data Tasks involved in the computation, suffering from the high cost of enclave creation on Intel SGX. This emphasizes the need for more advanced evaluation strategies.

Then, the performance comparison of the three strategies proposed in Chapter 5, namely Adaptive, Reverse-and-Replay and Repartition-and-Replay, demonstrates that the latter two are capable of carrying out computations with a maximal leakage protection while having practicable performances. Compared to Adaptive, they can handle a smaller leakage factor with better performances for the same number of objects.

Our experimental platform runs all the components of the PDMS on one machine. In order to cover the case of a PDMS leveraging remote Data Tasks, we simulated a higher cost of communication between the Core and the Data Tasks, similar to the average internet connection in France. By examining the behaviour of these strategies when confronted with higher communication costs, we showed that both Replay strategies have their areas of interest: Reverse-and-Replay is the most efficient when communications costs are similar to what can be expected on the same machine, while Repartition-and-Replay lessens the impact of higher communication costs (starting from a few hundreds microseconds).

Finally, we evaluated these strategies over a workload of queries and compared them with a lower bound in terms of performance (although unrealistic regarding our threat model). Reverse-and-Replay is less than 2 times more expensive than

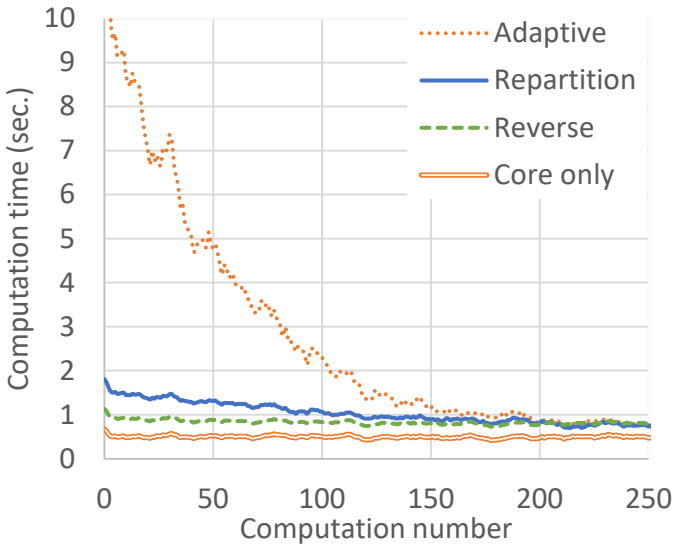


Figure 6.9: Workload 250 computations – GPS

this lower bound, supporting the idea that extensiveness and leakage protection are achieved at a low cost in a our architecture. Also, because of the Result Reuse mechanism all strategies have similar performances after several queries have been processed.

7 - Demonstration Prototype

Contents

In this chapter, we first describe the implementation details of the PDMS prototype used in our performance evaluations (Chapter 6). Then, we present the interactive games created to ease the comprehension of the security properties enforced by our architecture.

7.1 . ES-PDMS Prototype Implementation

We developed a Personal Data Management System prototype in C++ 17 using a Software Developement Kit named Open Enclave [112] v0.16.1. This SDK aims at generalizing the development of enclave applications across several technologies of Trusted Execution Environments by providing a hardware-agnostic open source library. In its current state, Open Enclave supports Intel SGX enclaves as well as a preview support for ARM TrustZone.

In its current state, our prototype uses Intel SGX enclaves and is composed of approximately 12.000 lines of code. The version of the prototype used to measure performances is accessible online[1]. Specifically, the following parts have been developed by Robin Carpentier:

- Handling computations carried out by the Core itself (See Section 6.2.5) or by splitting the input for several Data Tasks (See Section 4.3).
- Schemas and code to store and compute electric consumption and GPS data.
- Encryption/decryption of data exchanges between enclaves with AES GCM.
- Various optimizations and memory managements to handle the large datasets [199, 200] used in the performance evaluation.

The implementation of the main components are in common with the project developed by Floris Thiant.

[1] https://gitlab.inria.fr/rcarpent/es-pdms-prototype

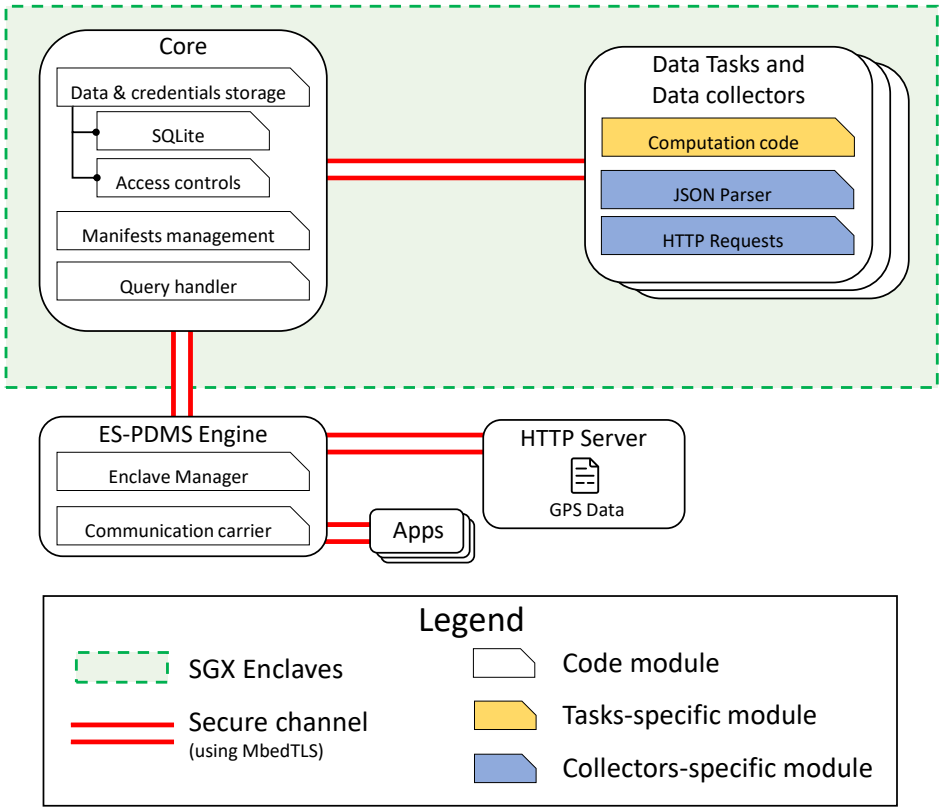


Figure 7.1: Components of the prototype.

7.1.1 . Implementation Components

Figure 7.1 shows the different components of our prototype.

ES-PDMS Engine. The ES-PDMS Engine is in charge of the life cycle of enclaves, creating and destroying them upon request by relying on the Open Enclave SDK. It also carries the communications between the different components: It receives the App's computation requests via sockets and forward them to the Core through a buffer; It transmits the HTTP requests and their results between the Core and the HTTP server. Finally, it dedicates threads to carry out the inter-enclaves communication via TLS packets on Circular Buffers.

Core. The Core runs inside an SGX enclave (a compatible CPU is needed). Its data management module is based on SQLite (v3) [203] which provides an SQL database engine self-embedded in a lightweight library written in C. Its communication module uses the Mbed TLS plugin [204] integrated in Open Enclave to establish secure channels with Data Tasks, remote parties or Apps. Intel SGX Remote Attestation with TLS [205] provides the way to authenticate enclaves via their measurement or their signer's identity in a standard TLS certificate. These measurements are loaded inside the Core via Manifests files (See Figure 7.2). The

```
{
    "identity" : {
        "name" : "CollectGPS",
        "signer" : "-----BEGIN PUBLIC KEY-----\n......",
        "type": "collect"
    },
    "storageAccesses" : [
        {
            "databaseName" : "UserDatabase",
            "table" : "GPSData",
            "columns" : [
                {
                    "name" : "timestamp"
                },
                {
                    "name" : "latitude"
                },
                {
                    "name" : "longitude"
                }
            ],
            "accessRight" : "insert"
        }
    ],
    "allowedConnections" : [
        {
            "hostname" : "localhost",
            "port" : "4433"
        }
    ]
}
```

Figure 7.2: Example of a Manifest file for the GPS Data Collector

serialization and deserialization of data necessary to send or receive data from Data Tasks is handled by FlatBuffers [206]. Its query handler module can manage several Data Tasks for the same computation, splitting the input data and/or receiving and propagating results from/to Data Tasks according to the computation's manifest.

Data Tasks and Data Collectors. Individual SGX enclaves are dedicated to each Data Task or Data Collector. In addition to their computation code, Data Tasks integrate the necessary modules to communicate securely with the Core. Furthermore, Data Collectors embed the necessary module to forge HTTP requests and parse their results. Both Data Tasks and Collectors have their identity and data rights registered in their Manifest.

Apps. Apps querying the PDMS are run on the same machine. They communicate with the Core through the ES-PDMS Engine via ZeroMQ [207] sockets.

HTTPS Server. A local HTTPS server is used to act as a service from which data will be collected (i.e. Google Maps in the *Green bonus* scenario). The server is loaded with a JSON file containing GPS trajectories sampled from the Geolife [200] Dataset. A Data Collector is granted the right to send HTTP requests, through the Core, to this server in its Manifest.

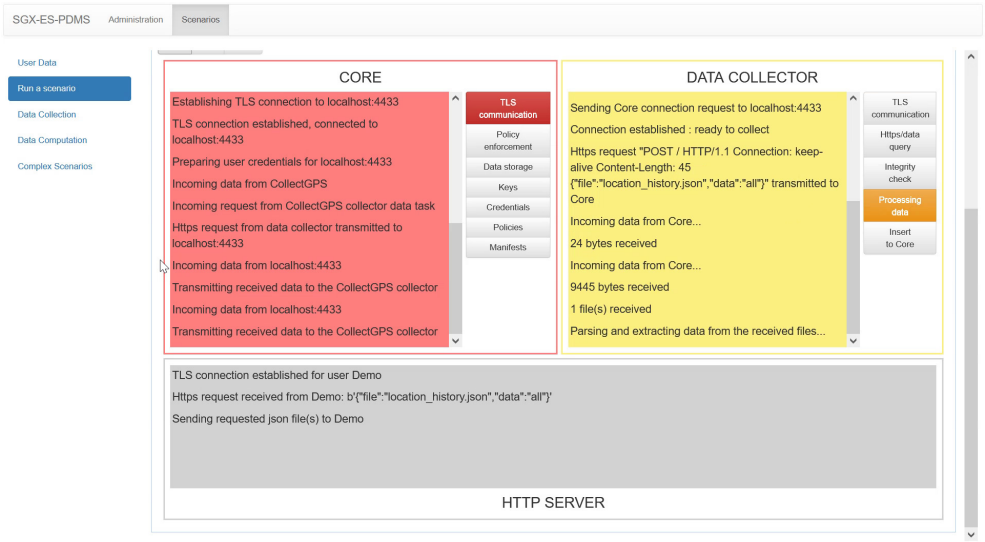


Figure 7.3: Demonstration interface of our PDMS prototype.

7.1.2 . Web Interface

A web interface (Figure 7.3) has been created to provide a visual display of the functioning of the prototype. It has been developed using Wt [208], a web framework for C++. This interface offers an easy way to start the different components of the prototype and launch execution scenarios. A collection or a computation task is selected by validating the corresponding manifest. Cleansed execution logs and graphical highlights are displayed during the execution to grasp the different interactions between the Core, the Data Tasks/Collectors and the third party, particularly emphasizing the security properties (Section 3.1.2).

Each security property (Section 3.1.2) is illustrated. Regarding data collection, after an initialization phase of the *enclaved Core/Data Tasks* (P1) and the establishment of *secure communications* (P2), we focus on traces related to *Core as proxy* (P3) and *Access control* (P4), allowing to manage the user's credentials at the Core level and preventing externalization of unauthorized personal to the third party. The data collected is stored in PDMS tables with appropriate permissions for further use (including by the compute tasks specified in the manifests). During the Compute step, attention is drawn to the traces related to *Access Control* (P4) and *Attestations* (P5). The interface shows both the secure workflow applied on the inputs (e.g., provenance, integrity, etc.) and the use of SGX attestation enabling the Core to attest at run time the compliance of the final result with the execution manifest.

In its current state, this interface can launch the collection and computation tasks of the *Green bonus* scenario.

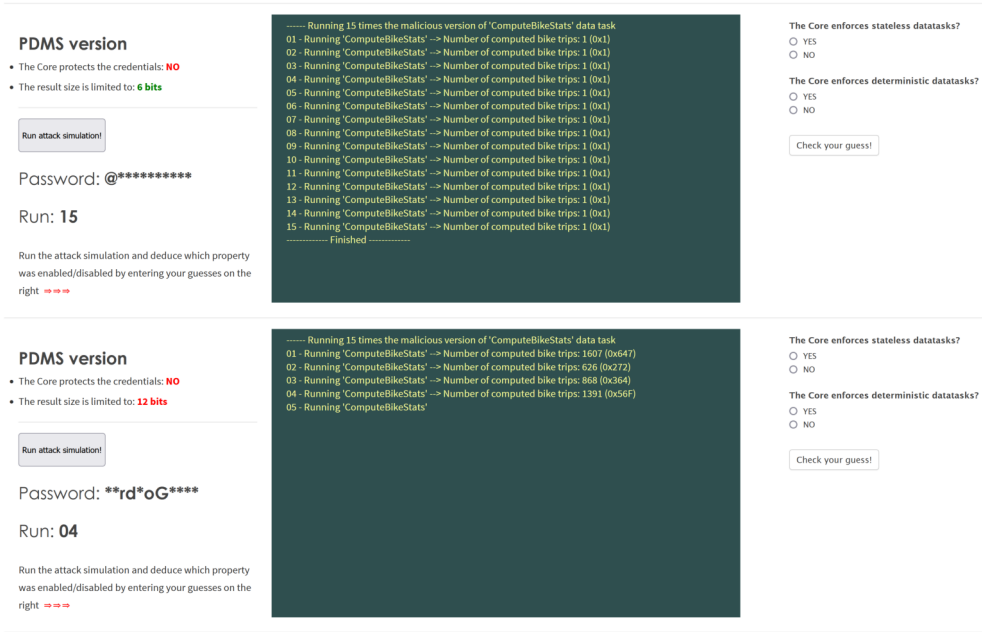


Figure 7.4: Game 2.

7.2 . Interactive Games

In order to help the understanding of the security properties of the architecture, three games focused on the *Green bonus* scenario have been made accessible online[2] ans shown in Figure 7.4 and 7.5. These games involve theoretical attacks on the PDMS from either a malicious PDMS user or a malicious PDMS querier.

1st Game. The first game supposes that an attack has been conducted by a PDMS user –an employee in the *Green bonus* scenario–. They falsify their GPS data to get the maximum number of bike trips hoping to increase the financial benefit obtained. To do so, the employee allegedly updates the location history recovered from Google Maps, before it is taken as input by the compute Task. Two security properties of the PDMS are either enabled or disabled and the player must guess the impact of the attack on the final result of the computation of the number of bike trips. Educational execution logs are provided to help picturing what would be the main steps of a real execution.

The next two games consider attacks conducted by PDMS queriers –an employer– wishing to gain access to raw personal data of the PDMS user, using a legitimate granted execute privilege on manifests leveraging potentially corrupted Data Tasks –their code being potentially produced by the querier itself–.

2nd Game. We first consider a corrupted compute Task used by the employer to leak sensitive information of small size (such as a password), to demonstrate the importance of limiting the result size and enforcing stateless and deterministic

[2] https://project.inria.fr/espdms

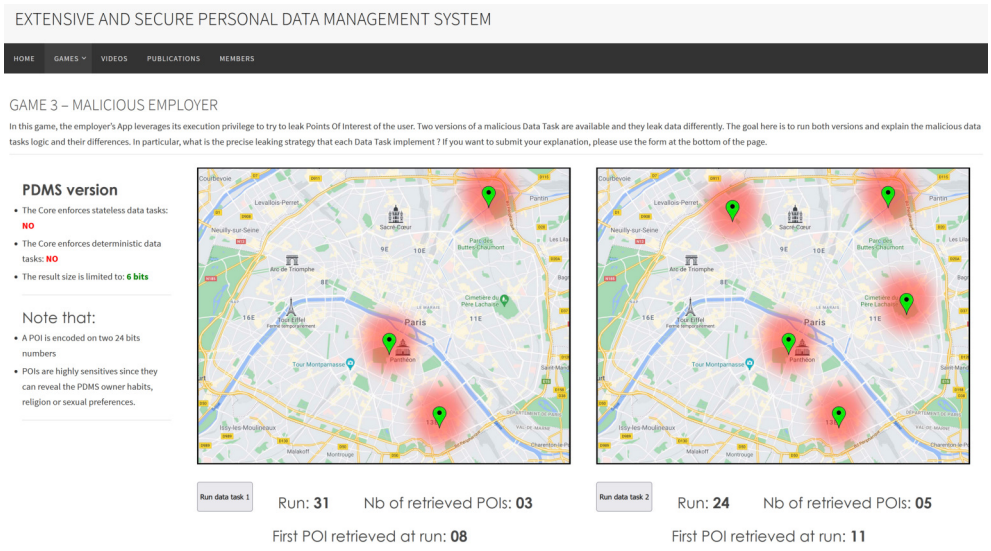


Figure 7.5: Game 3.

Data Tasks. The attack simulation produces educational execution logs and reveals some characters of the password obtained by the attacker. The player must then guess if stateless and/or deterministic Data Tasks were enforced.

3rd Game. The third attack aims to leak larger sets of personal data, potentially all personal data accessible to the corrupted Task. The attacker is the employer trying to exploit a corrupted task to gain access to the detailed location history of employees. Two different Data Tasks incorporating two effective methods for leaking points of interest (POIs) are presented. The player can run the scenario with each of the two Data Tasks, and is then challenged to identify which leaking methods was implemented is each of them. A final challenge for the player is to find the minimum number of executions needed to reveal a given amount of points of interests in the PDMS user's GPS history, by exploiting the best leaking method given stateless and non deterministic Data Tasks.

7.3 . Conclusion

In this chapter we described the Personal Data Management System prototype that we implemented and used to measure the performances of our proposals. It uses Intel SGX, a concrete instance of a Trusted Execution Environment embedded inside Intel CPUs (see Section 2.2) to perform computations with confidentiality and attestation guarantees. This prototype is a significant step in demonstrating the feasibility of a fully-featured Extensive and Secure PDMS architecture. Many development challenges still need to be addressed. For example, the integration of a Sandbox mechanism (See Section 3.1.2) for the Data Tasks and Data Collectors to prevent voluntary leak. Also, the use of SGX sealing capacities to ensure a

persistent storage between reboots of the Core or the operating system.

This chapter also described three educational games created to clarify the benefits of the proposed security properties to defend against attacks, either from a malicious PDMS user or a malicious PDMS querier.

8 - Conclusion

Nowadays, the privacy of individuals is under constant assault. Gathering data about people is a gigantic business involving some of the most prominent companies in the world. As individuals, this widespread collection of data about themselves can feel overwhelming and privacy can be thought of as either constraining or time-consuming, especially for non-experts. Thus, it is crucial to bring privacy-preserving solutions designed for individuals to collect, store and easily benefit from their data. Personal Data Management Systems (PDMS), or Personal Clouds, seek to fulfill this role by empowering users to regain control of their data. This thesis is built upon the PDMS paradigm and proposes solutions to securely address a specific yet frequent use-case of the PDMS: computing user's data on behalf of a third party having a legitimate purpose of receiving the result of this computation.

8.1 . Contributions

Two research questions (Section 3.3) defined the main goal of this thesis: (1) Is there an upper bound on the potential leakage of personal information that can be guaranteed to the PDMS user, when successively evaluating a user-defined function on large sensitive data sets, under the considered PDMS architecture, computation and threat models? (2) Is there a performance-acceptable execution strategy guaranteeing minimal leakage with potentially large volumes of personal data?

Our contributions and answers to these questions are as followed :

1. We designed a concrete architecture for an Extensive and Secure Personal Data Management System (ES-PDMS), tailored to the specifics of a popular Trusted Execution Environment (TEE) named Intel SGX. Our architecture instantiates and extends some of the security properties introduced in [23]. The security of this design is built upon the security properties of SGX: code integrity, data integrity and confidentiality, and attestation. Extensiveness emerges from the inclusion of potentially untrusted third-party computation codes used to process data, detached from the trusted Core of the PDMS engine. We defined a computation model where a third party can receive legitimate aggregate results. We consider a powerful attacker who (i) programs the computation code, (ii) chooses the subset of authorized data that will be computed upon each request and (iii) has the granted privilege of requesting as many computation as they want, receiving the associated results. Using those, they will try to leak data out of the PDMS. The architecture, the computation model and the threat model are presented in Chapter 3.

   From this architecture, we proposed a first set of countermeasures to reduce

leakage, namely our *security building blocks*. These three building blocks take the form of design rules to be enforced during computations to defend against various data leaking strategies that can be implemented inside the computation code by the attacker. The first one enforces a *stateless* property guaranteeing that data from past computations cannot affect new computations. The second one prevents accesses to usual sources of randomness, only allowing chosen randoms, making each computation deterministic and reproducible. The last building block partitions the input of a computation and distributes each part to one computation node, fixing the quantity of data that can be leaked about a particular object. These three building blocks have an important impact on the leakage: from the potentially limitless leakage arising from our threat model, their combination enforces an *upper bound* on the amount of data leakage an attacker can receive. Thus, a positive answer can be given to our first research question.

Chapter 4 is dedicated to these building blocks. They have been previously sketched in a poster-paper [24] published at the 6th IEEE European Symposium on Security and Privacy (EuroS&P 2021), then described in detail in a position paper [25] published at the 11th International Conference on Data Science, Technology and Applications (DATA 2022).

2. Furthermore, we designed *execution strategies* from these building blocks, protecting against data leakage with a reasonable performance overhead. As their main components, these strategies use result storage, re-partition of the input data and replay of the computations to efficiently impose the aforementioned upper bound on data leakage. Each strategy also offers a tradeoff between leakage protection and performances, with tunable parameters for each computation. Moreover, our proposal covers various contexts, one of the strategy being suitable when the latency between components of the architecture is low (e.g. on the same machine), whereas another is better than the first when high latency is involved (e.g. network latency between remote machines). These strategies offer a valid answer to our second research question.

   These strategies are detailed in Chapter 5. Together with the security building blocks, they are part of a research paper [26] published at the 34th International Conference on Scientific and Statistical Database Management (SSDBM 2022) and also presented at the 38th French Conference on Data Management - Principles, Technologies and Applications (BDA 2022) [27].

3. Finally, we implemented a prototype of an ES-PDMS in C++ effectively using Intel SGX. This open-source prototype is a direct adaptation of our architecture design. It includes the code to run our main use-case scenario entirely, namely the *Green bonus*, by collecting and computing GPS data. The prototype also contains a web interface to facilitate the visualisation

of the computations. This implementation has also been used to evaluate the impact on performances of our *security building blocks* and *execution strategies*. We show that the security overhead remains reasonable compared to the benefits in privacy it offers. Besides, we created educational games to improve the comprehension of our proposal.

The performances are described in Chapter 6 and were published alongside the execution strategies in [26]. Chapter 7 describes the prototype and the educational games, which were part of a demonstration paper [28] published at the 25th International Conference on Extending Database Technology (EDBT 2022) and also presented at the 2022 CNIL's International Privacy Research Day [29] and at the 38th French Conference on Data Management - Principles, Technologies and Applications (BDA 2022) [30].

8.2 . Perspectives

To the best of our knowledge, this thesis is the first contribution to address the problem of data leakage through results on an ES-PDMS. Thereby, further challenges need to be studied to offer a fully-featured ES-PDMS.

On the one hand, concrete challenges can be derived from this thesis.

- First, to reduce the potential data leakage, complementary security mechanisms can be employed for some Apps, e.g., imposing a query budget, restricting the σ predicates. Defining such restrictions and incorporating them into App manifests would definitely make sense, but it is left as future work, as we wanted to be generic with regard to the application types and studied the worst-case scenarios.

- Also, aggregate computations are generally basic and as such could be computed by the Core. The computation of agg by the Core introduces an additional trust assumption which could help to further reduce the potential data leakages.

- To our knowledge, despite the benefits it offers, no Personal Cloud Company is actively using Intel SGX as a way so secure the storage and computation of their users' data. An engaging future work would be to collaborate with such company to transfer our work, potentially by taking our PDMS prototype to a next step for it to become a stable ES-PDMS product usable by individuals.

On the other hand, important research challenges are also at stake.

- In our computation model, the successful running of a computation is dependant on the ability of the Core to evaluate the σ selection predicates sent by App for each request. This is a reasonable assumption if basic predicates are considered over metadata associated with objects (e.g., temporal,

file/object type or size, tags). However, because of the Core minimality, it is not reasonable to assume that it will support more complex selection predicates. For example, the Core cannot select GPS trajectories based on whether they contain a specific point of interest or not. Besides, advanced selection would require specific data indexing to be efficient. Considering our architecture, only a code loaded as a Data Task can be used to build such index. Yet, the introduction of an untrusted code to build an index used to select the input of computations jeopardises our security solutions. Indeed, this code should legitimately have access to the entire dataset it has to index and could thus introduce a flaw in said index based on the content of the dataset. For example, it could build the index such that an index lookup results in the selection of objects whose first bits, taken together, corresponds to the content of an entire object. As such, by using a straightforward approach, the upper bound on the leakage of data defined in Chapter 4 does not hold anymore. New solutions have to be designed to maintain this bound. For instance, the Core could check the validity of index built by picking up subsets of the dataset and trying to replicate the result, reusing the concept of replay (Section 5.3). A complementary issue concerns the handling of the index lookup. Can the Core perform this task or can we rely on another Data Task?

- Another interesting challenge would be to add the support of more advanced calculations to cope with broader users' needs. Some types of computations need parameters given at execution time (e.g., a similarity function for time series or images receiving the object to be compared as an input parameter from App). Such parameters cannot be handled by the Core as they are needed to compute the content of the objects, which the Core cannot interpret. However, offering a direct access of these parameters to cmp Data Tasks could threaten the security of our solutions. Indeed, two computations of the same subset of objects with two different parameter's value enable a corrupted Data Task to leak two distinct parts of each object, effectively overstepping the leakage upper bound built in Chapter 4.

- Our study is focused on tackling leakage during computations of the data of a single individual on its own PDMS. One promising perspective that can be envisioned is the support of computations over a distributed set of users. Several PDMS users may want to be involved in a collective processing of their data, for example to compute statistics of energy consumption of an entire city district. While our solutions can be applied on each involved individual, it falls short of providing strong guarantees on the distribution of potential leakage among users. For instance, in the case of computing the average energy consumption of multiple users over a period of time, each user can compute their own average on their PDMS using our solutions.

Then, one PDMS can receive all the averages and compute the final result. In the case of malicious Data Tasks, the final result can contain a leak of data belonging to only one user. Thus, proper strategies have to be designed to balance the potential leakage of data among participants of a collective processing.

A - Appendix - Repartition-and-Replay - Derivative analysis

In the Repartition-and-Replay strategy, the total number of Data Tasks used across all replay levels depends on the number of partitions m, the number of input objects n and the selected leakage factor k. When choosing the minimal leakage factor $k = 1$, the number of Data Tasks is given by the following function:

$$f(m) = m \cdot log_m(n)$$

The domain of m is $\mathbb{N}^* \setminus \{1\}$. Indeed, m is a number of partitions (thus a positive integer) and the Repartition-and-Replay strategy requires at least two partitions, otherwise all objects are computed together and nothing can be achieved by replaying such computation. The value for m leading to the least number of Data Tasks can be found by analysing the derivative of f.

$$\begin{aligned}
f'(m) &= \frac{d}{dm}\left(m \cdot log_m(n)\right) \\
f'(m) &= m \cdot \frac{d}{dm} log_m(n) + log_m(n) \cdot \frac{d}{dm} m \\
f'(m) &= m \cdot \frac{d}{dm} \frac{ln(n)}{ln(m)} + log_m(n) \\
f'(m) &= m \cdot ln(n) \cdot \frac{d}{dm} \frac{1}{ln(m)} + log_m(n) \\
f'(m) &= m \cdot ln(n) \cdot \frac{-\frac{1}{m}}{ln(m)^2} + log_m(n) \\
f'(m) &= ln(n) \cdot \frac{-1}{ln(m)^2} + log_m(n) \\
f'(m) &= \frac{-ln(n)}{ln(m)^2} + \frac{ln(n) \cdot ln(m)}{ln(m)^2} \\
f'(m) &= \frac{ln(n) \cdot (ln(m) - 1)}{ln(m)^2}
\end{aligned}$$

To find the least value of f we need to solve $f'(m) = 0$. Given that n is a number of objects, we are interested in cases where $n \in \mathbb{N}^* \setminus \{1\}$, because in the case of $n = 1$ there is no batch to make and one Data Task is sufficient. Considering f', $ln(m)^2$ and $ln(n)$ are always positive and not null for $m, n \in \mathbb{N}^* \setminus \{1\}$. Thus the logical equivalence:

$$f'(m) = 0 \iff \frac{ln(n) \cdot (ln(m) - 1)}{ln(m)^2} = 0 \iff ln(m) - 1 = 0 \iff m = e$$

where e is Euler's number. The closest integer leading to the least number of Data Tasks is $m = 3$.

Bibliography

[1] David Reinsel, John Gantz, and John Rydning. *The digitization of the world from edge to core*. Tech. rep. International Data Corporation, 2018. url: https://perma.cc/N3FW-WRRU.

[2] Cisco. *Cisco Annual Internet Report (2018–2023)*. Tech. rep. 2020. url: https://perma.cc/ZKW6-UZMJ.

[3] European Council. "Regulation EU 2016/679 of the European Parliament and of the Council." In: *Official Journal of the European Union (OJ)* 59.1-88 (2016), p. 294. url: http://data.europa.eu/eli/reg/2016/679 (visited on 2022-10-12).

[4] Neal Lathia and Licia Capra. "Mining Mobility Data to Minimise Travellers' Spending on Public Transport." In: *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '11. 2011, pp. 1181–1189. doi: 10.1145/2020408.2020590.

[5] Mostafa Amin-Naseri, Pranamesh Chakraborty, Anuj Sharma, Stephen B Gilbert, and Mingyi Hong. "Evaluating the reliability, coverage, and added value of crowdsourced traffic incident reports from Waze." In: *Transportation research record* 2672.43 (2018), pp. 34–43. doi: 10.1177/0361198118790619.

[6] Yoko Brigitte Wang, Amanda J Page, Tiffany K Gill, and Yohannes Adama Melaku. "Association of dietary and nutrient patterns with systemic inflammation in community dwelling adults." In: (2022). doi: 10.3389/fnut.2022.977029.

[7] Roman Pabayo, John C. Spence, Linda Casey, and Kate Storey. "Food Consumption Patterns: In Preschool Children." In: *Canadian Journal of Dietetic Practice and Research* 73.2 (2012), pp. 66–71. doi: 10.3148/73.2.2012.66.

[8] Christoph Becker, Isadora Kirchmaier, and Stefan T Trautmann. "Marriage, parenthood and social network: Subjective well-being and mental health in old age." In: *PloS one* 14.7 (2019). doi: 10.1371/journal.pone.0218704.

[9] M. Mazhar Rathore, Awais Ahmad, Anand Paul, and Seungmin Rho. "Urban planning and building smart cities based on the Internet of Things using Big Data analytics." In: *Computer Networks* 101 (2016), pp. 63–80. doi: 10.1016/j.comnet.2015.12.023.

[10] Kaile Zhou and Shanlin Yang. “Understanding household energy consumption behavior: The contribution of energy big data analytics.” In: 56 (2016), pp. 810–819. doi: 10.1016/j.rser.2015.12.001.

[11] Thomas Niebel, Fabienne Rasel, and Steffen Viete. “BIG data – BIG gains? Understanding the link between big data analytics and innovation.” In: *Economics of Innovation and New Technology* 28.3 (2019), pp. 296–316. doi: 10.1080/10438599.2018.1493075.

[12] *Firefox Monitor - Recent data breaches*. url: https://monitor.firefox.com/breaches (visited on 2022-10-03).

[13] S. Zuboff. *The Age of Surveillance Capitalism: The Fight for a Human Future at the New Frontier of Power: Barack Obama's Books of 2019*. 2019. isbn: 978-1-78283-274-4.

[14] Laura Poitras. “Citizenfour.” In: *Lectures, publications reçues* (2015).

[15] Wolfie Christl, Katharina Kopp, and Patrick Urs Riechert. *Corporate surveillance in everyday life*. Tech. rep. Cracked Labs, 2017. url: https://perma.cc/84TD-5RV5.

[16] KPMG. *Corporate data responsibility: Bridging the consumer trust gap*. 2021. url: https://perma.cc/L7YT-C96Y.

[17] California State Legislature. *California Consumer Privacy Act (CCPA)*. 2018.

[18] *California Proposition 24 (2020) - California Privacy Rights Act (CPRA)*. 2020.

[19] *Signal - Private Messenger*. url: https://signal.org/ (visited on 2022-10-05).

[20] *Proton - Your Online Privacy, by Default*. url: https://proton.me/ (visited on 2022-10-05).

[21] *Framasoft - Change the world one byte at a time*. url: https://framasoft.org/en/ (visited on 2022-10-05).

[22] Cozy. *Cozy Cloud*. url: https://cozy.io (visited on 2022-09-27).

[23] Nicolas Anciaux, Philippe Bonnet, Luc Bouganim, Benjamin Nguyen, Philippe Pucheral, Iulian Sandu Popa, and Guillaume Scerri. “Personal data management systems: The security and functionality standpoint.” In: *Information Systems* 80 (2019), pp. 13–35. doi: 10.1016/j.is.2018.09.002. url: https://hal.inria.fr/hal-01898705.

[24] Robin Carpentier, Iulian Sandu Popa, and Nicolas Anciaux. “Poster: Reducing Data Leakage on Personal Data Management Systems.” In: *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*. Los Alamitos, CA, USA: IEEE Computer Society, Sept. 2021, pp. 716–718. doi: 10.1109/EuroSP51992.2021.00057. url: https://hal.inria.fr/hal-03536381.

[25] Robin Carpentier, Iulian Sandu Popa, and Nicolas Anciaux. “Local Personal Data Processing with Third Party Code and Bounded Leakage.” In: *Proceedings of the 11th International Conference on Data Science, Technology and Applications - DATA*. 2022, pp. 520–527. isbn: 978-989-758-583-8. doi: 10.5220/0011321900003269. url: https://hal.inria.fr/hal-03686098.

[26] Robin Carpentier, Iulian Sandu Popa, and Nicolas Anciaux. “Data Leakage Mitigation of User-Defined Functions on Secure Personal Data Management Systems.” In: *34th International Conference on Scientific and Statistical Database Management*. SSDBM 2022. 2022. isbn: 978-1-4503-9667-7. doi: 10.1145/3538712.3538741. url: https://hal.inria.fr/hal-03692175.

[27] Robin Carpentier, Iulian Sandu Popa, and Nicolas Anciaux. *Data Leakage Mitigation of User-Defined Functions on Secure Personal Data Management Systems*. Presentation at the 38th French Conference on Data Management - Principles, Technologies and Application. Clermont-Ferrand, France, Oct. 2022.

[28] Robin Carpentier, Floris Thiant, Iulian Sandu Popa, Nicolas Anciaux, and Luc Bouganim. “An Extensive and Secure Personal Data Management System Using SGX.” In: *Proceedings of the 25th International Conference on Extending Database Technology, EDBT*. 2022, 2:570–2:573. doi: 10.48786/edbt.2022.53. url: https://hal.inria.fr/hal-03580286.

[29] Robin Carpentier, Floris Thiant, Iulian Sandu Popa, Nicolas Anciaux, and Luc Bouganim. *Outline: An Extensive and Secure Personal Data Management System Using SGX*. Presentation at the 1st CNIL International Privacy Research Day. Commission Nationale de l’Informatique et des Libertés, Paris, France, June 2022. url: https://hal.inria.fr/hal-03763815.

[30] Robin Carpentier, Floris Thiant, Iulian Sandu Popa, Nicolas Anciaux, and Luc Bouganim. *An Extensive and Secure Personal Data Management System Using SGX*. Presentation at the 38th French Conference on Data Management - Principles, Technologies and Application. Clermont-Ferrand, France, Oct. 2022.

[31] Minna M Rantanen and Jani Koskinen. “Respecting the individuals of data economy ecosystems.” In: *International Conference on Well-Being in the Information Society*. Springer. 2020, pp. 185–196. doi: 10.1007/978-3-030-57847-3_13.

[32] Andrei Vlad Sambra, Essam Mansour, Sandro Hawke, Maged Zereba, Nicola Greco, Abdurrahman Ghanem, Dmitri Zagidulin, Ashraf Aboulnaga, and Tim Berners-Lee. *Solid: A Platform for Decentralized Social Applications Based on Linked Data*. Tech. rep. MIT CSAIL & Qatar Computing Research Institute, 2016.

[33] Sofie Verbrugge, Frederic Vannieuwenborg, Marlies Van der Wee, Didier Colle, Ruben Taelman, and Ruben Verborgh. "Towards a personal data vault society: an interplay between technological and business perspectives." In: *2021 60th FITCE Communication Days Congress for ICT Professionals: Industrial Data – Cloud, Low Latency and Privacy (FITCE)*. 2021, pp. 1–6. doi: 10.1109/FITCE53297.2021.9588540.

[34] Seong-hyun Min and Kyung-ho Son. "MyData Personal Data Store Model (PDS) to Enhance Information Security for Guarantee the Self-determination rights." In: *KSII Transactions on Internet and Information Systems (TIIS)* 16.2 (2022), pp. 587–608.

[35] *Google Drive*. url: https://drive.google.com (visited on 2022-09-27).

[36] *Dropbox*. url: https://www.dropbox.com (visited on 2022-09-27).

[37] *OneDrive*. url: https://onedrive.com (visited on 2022-09-27).

[38] *iCloud*. url: https://www.icloud.com (visited on 2022-09-27).

[39] *IDrive*. url: https://www.idrive.com (visited on 2022-09-27).

[40] *Backblaze*. url: https://www.backblaze.com (visited on 2022-09-27).

[41] *SugarSync*. url: https://www1.sugarsync.com (visited on 2022-09-27).

[42] *livedrive*. url: https://www2.livedrive.com (visited on 2022-09-27).

[43] *Jottacloud*. url: https://www.jottacloud.com (visited on 2022-09-27).

[44] *Mega*. url: https://mega.nz (visited on 2022-09-27).

[45] *Proton Drive*. url: https://proton.me/drive (visited on 2022-09-27).

[46] *SpiderOak*. url: https://spideroak.com (visited on 2022-09-26).

[47] *NordLocker*. url: https://nordlocker.com (visited on 2022-09-27).

[48] *pCloud*. url: https://www.pcloud.com (visited on 2022-09-27).

[49] *tresorit*. url: https://tresorit.com (visited on 2022-09-27).

[50] *sync.com*. url: https://www.sync.com (visited on 2022-09-27).

[51] *Filen*. url: https://filen.io (visited on 2022-09-27).

[52] *Koofr*. url: https://koofr.eu (visited on 2022-09-27).

[53] *Digi.me*. url: https://digi.me (visited on 2022-09-27).

[54] *Solid*. url: https://solidproject.org/ (visited on 2022-09-30).

[55] *BitsAbout.me*. url: https://bitsabout.me (visited on 2022-09-27).

[56] *Prifina*. url: https://www.prifina.com (visited on 2022-09-28).

[57] *mydex*. 2007. url: https://mydex.org/ (visited on 2021-11-10).

[58] *Nextcloud*. url: https://nextcloud.com (visited on 2022-09-24).

[59] *MyDataMood*. url: https://mydatamood.com (visited on 2022-09-28).

[60] Yves-Alexandre de Montjoye, Erez Shmueli, Samuel S. Wang, and Alex Sandy Pentland. "openPDS: Protecting the Privacy of Metadata through SafeAnswers." In: *PLOS ONE* 9.7 (2014), pp. 1–9. doi: 10.1371/journal.pone.0098790.

[61] Amir Chaudhry, Jon Crowcroft, Heidi Howard, Anil Madhavapeddy, Richard Mortier, Hamed Haddadi, and Derek McAuley. "Personal Data: Thinking Inside the Box." In: *Aarhus Series on Human Centered Computing* 1.1 (2015), p. 4. doi: 10.7146/aahcc.v1i1.21312.

[62] *Personium*. url: https://personium.io (visited on 2021-11-11).

[63] *amber X*. url: https://www.myamberlife.com (visited on 2022-09-28).

[64] *CloudLocker*. url: https://web.archive.org/web/20220308035445/https://www.cloudlocker.eu/en/index.html (visited on 2022-09-27).

[65] *Helixee*. url: http://www.helixee.me/home (visited on 2022-09-27).

[66] *Meet Lima*. url: https://meetlima.com (visited on 2022-09-27).

[67] Tristan Allard, Nicolas Anciaux, Luc Bouganim, Yanli Guo, Lionel Le Folgoc, Benjamin Nguyen, Philippe Pucheral, Indrajit Ray, Indrakshi Ray, and Shaoyi Yin. "Secure personal data servers: a vision paper." In: *Proceedings of the VLDB Endowment* 3.1-2 (2010), pp. 25–35. doi: 10.14778/1920841.1920850. url: https://hal.inria.fr/inria-00551875/.

[68] Nicolas Anciaux, Philippe Bonnet, Luc Bouganim, Benjamin Nguyen, Iulian Sandu Popa, and Philippe Pucheral. "Trusted Cells : A Sea Change for Personnal Data Services." In: *Proceedings of the 6th biennal Conference on Innovative Database Research (CIDR 2013)*. 2013. url: https://hal.inria.fr/hal-00768379.

[69] Nicolas Anciaux, Mehdi Benzine, Luc Bouganim, Kévin Jacquemin, Philippe Pucheral, and Shaoyi Yin. "Restoring the Patient Control over Her Medical History." In: *2008 21st IEEE International Symposium on Computer-Based Medical Systems*. 2008, pp. 132–137. doi: 10.1109/CBMS.2008.101.

[70] Nicolas Anciaux et al. "A Tamper-Resistant and Portable Healthcare Folder." In: *International Journal of Telemedicine and Applications* 2008 (2008). doi: 10.1155/2008/763534.

[71] Nicolas Anciaux, Sébastien Guillotton, Luc Bouganim, Sergio Ilarri, Alain Kamgang, Abraham Ngami, Christophe Nouedoui, Philippe Pucheral, and Maurice Tchuente. "Managing Personal Health Records in an Infrastructure-Weak Environment." In: *e-Infrastructure and e-Services*. 2016, pp. 178–191. doi: 10.1007/978-3-319-43696-8_18.

[72] Saliha Lallali, Nicolas Anciaux, Iulian Sandu Popa, and Philippe Pucheral. "Supporting secure keyword search in the personal cloud." In: *Information Systems* 72 (2017). doi: 10.1016/j.is.2017.09.003.

[73] Nicolas Anciaux, Benjamin Nguyen, and Iulian Sandu-Popa. "Personal Data Management with Secure Hardware: How to keep your Data at Hand." In: *IEEE 14th International Conference on Mobile Data Management*. Vol. 2. Milan, Italy, June 2013, pp. 1–2. url: https://hal.archives-ouvertes.fr/hal-00935613.

[74] Nicolas Anciaux, Benjamin Nguyen, and Iulian Sandu-Popa. "Tutorial: Managing Personal Data with Strong Privacy Guarantees." In: *17th International Conference on Extending Database Technology (EDBT)*. 2014, pp. 672–673. doi: 10.5441/002/edbt.2014.71.

[75] Nicolas Anciaux, Luc Bouganim, Philippe Pucheral, Yanli Guo, Lionel Le Folgoc, and Shaoyi Yin. "MILo-DB: a personal, secure and portable database machine." In: *Distributed and Parallel Databases* 32.1 (2014), pp. 37–63. issn: 1573-7578. doi: 10.1007/s10619-012-7119-x.

[76] Paul Tran-Van, Nicolas Anciaux, and Philippe Pucheral. "SWYSWYK: A privacy-by-design paradigm for personal information management systems." In: *International Conference on Information Systems Development (ISD)*. 2017. url: https://hal.inria.fr/hal-01675090/.

[77] Cuong Quoc To, Benjamin Nguyen, and Philippe Pucheral. "Privacy-Preserving Query Execution using a Decentralized Architecture and Tamper Resistant Hardware." In: *17th International Conference on Extending Database Technology (EDBT)*. 2014. doi: 10.5441/002/edbt.2014.44.

[78] Quoc-Cuong To, Benjamin Nguyen, and Philippe Pucheral. "Private and Scalable Execution of SQL Aggregates on a Secure Decentralized Architecture." In: *ACM Trans. Database Syst.* 41.3 (2016). doi: 10.1145/2894750.

[79] Dai Hai Ton That, Iulian Sandu Popa, Karine Zeitouni, and Cristian Borcea. "PAMPAS: Privacy-Aware Mobile Participatory Sensing Using Secure Probes." In: *Proceedings of the 28th International Conference on Scientific and Statistical Database Management*. SSDBM '16. 2016. doi: 10.1145/2949689.2949704.

[80] Nicolas Anciaux et al. *PlugDB*. url: https://project.inria.fr/plugdb/en/ (visited on 2022-10-07).

[81] Nicolas Anciaux, Saliha Lallali, Iulian Sandu Popa, and Philippe Pucheral. "A Scalable Search Engine for Mass Storage Smart Objects." In: *Proc. VLDB Endow.* 8.9 (2015), pp. 910–921. doi: 10.14778/2777598.2777600.

[82] Tristan Allard, Benjamin Nguyen, and Philippe Pucheral. "MetaP: Revisiting Privacy-Preserving Data Publishing using Secure Devices." In: *Distributed and Parallel Databases* (2014), pp. 1–55. doi: 10.1007/s10619-013-7122-x. url: https://hal.archives-ouvertes.fr/hal-00934586.

[83] N. Anciaux, L. Bouganim, T. Delot, S. Ilarri, L. Kloul, N. Mitton, and P. Pucheral. "Folk-IS: Opportunistic Data Services in Least Developed Countries." In: *Proc. VLDB Endow.* 7.5 (2014), pp. 425–428. doi: 10.14778/2732269.2732278.

[84] Philippe Pucheral, Luc Bouganim, Patrick Valduriez, and Christophe Bobineau. "PicoDBMS: Scaling down database techniques for the smartcard." In: *The VLDB Journal* 10.2 (Sept. 2001), pp. 120–132. doi: 10.1007/s007780100047.

[85] Saliha Lallali, Nicolas Anciaux, Iulian Sandu Popa, and Philippe Pucheral. "A Secure Search Engine for the Personal Cloud." In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. SIGMOD '15. 2015, pp. 1445–1450. doi: 10.1145/2723372.2735376.

[86] Nicolas Anciaux, Luc Bouganim, Yanli Guo, Philippe Pucheral, Jean-Jacques Vandewalle, and Shaoyi Yin. "Pluggable Personal Data Servers." In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*. SIGMOD '10. 2010, pp. 1235–1238. doi: 10.1145/1807167.1807328.

[87] Susanta Mitra and Somsubhra Gupta. "Mobile learning under personal cloud with a virtualization framework for outcome based education." In: *Education and Information Technologies* 25.3 (2020), pp. 2129–2156. doi: 10.1007/s10639-019-10043-z.

[88] Mahmuda Akter, Abdullah Gani, Md. Obaidur Rahman, Mohammad Mehedi Hassan, Ahmad Almogren, and Shafiq Ahmad. "Performance Analysis of Personal Cloud Storage Services for Mobile Multimedia Health Record Management." In: *IEEE Access* 6 (2018), pp. 52625–52638. doi: 10.1109/ACCESS.2018.2869848.

[89] Charith Perera, Susan Wakenshaw, Tim Baarslag, Hamed Haddadi, Arosha Bandara, Richard Mortier, Andy Crabtree, Irene Ng, Derek McAuley, and Jon Crowcroft. *Valorising the IoT Databox: Creating Value for Everyone*. 2016. doi: 10.48550/ARXIV.1609.03312.

[90] Christian Meurisch, Bekir Bayrak, and Max Mühlhäuser. "Privacy-Preserving AI Services Through Data Decentralization." In: *Proceedings of The Web Conference 2020*. WWW '20. New York, NY, USA, 2020, pp. 190–200. isbn: 978-1-4503-7023-3. doi: 10.1145/3366423.3380106.

[91] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. "Trusted Execution Environment: What It is, and What It is Not." en. In: *2015 IEEE Trustcom/BigDataSE/ISPA*. IEEE, Aug. 2015, pp. 57–64. doi: 10.1109/Trustcom.2015.357.

[92] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R Savagaonkar. *Innovative instructions and software model for isolated execution.* Intel Corporation, 2013. url: https://perma.cc/B93F-JSTY.

[93] Victor Costan and Srinivas Devadas. *Intel SGX Explained*. Cryptology ePrint Archive, Paper 2016/086. 2016. url: https://eprint.iacr.org/2016/086.

[94] ARM. *Building a Secure System using TrustZone Technology*. 2008. url: https://perma.cc/WE2E-WV2A.

[95] Sandro Pinto and Nuno Santos. "Demystifying Arm TrustZone: A Comprehensive Survey." In: *ACM Comput. Surv.* 51.6 (2019). doi: 10.1145/3291047.

[96] S. Gueron. "Memory Encryption for General-Purpose Processors." In: *IEEE Security Privacy* 14.6 (2016), pp. 54–62. doi: 10.1109/MSP.2016.124.

[97] Ittai Anati, Shay Gueron, Simon P Johnson, and Vincent R Scarlata. *Innovative Technology for CPU Based Attestation and Sealing*. Intel Corporation, 2013. url: https://perma.cc/CX5W-D56Z.

[98] Ernie Brickell and Jiangtao Li. "Enhanced privacy ID from bilinear pairing for hardware authentication and attestation." In: *2010 IEEE Second International Conference on Social Computing*. IEEE, 2010, pp. 768–775. doi: 10.1504/IJIPSI.2011.043729.

[99] Jaehyuk Lee, Jinsoo Jang, Yeongjin Jang, Nohyun Kwak, Yeseul Choi, Changho Choi, Taesoo Kim, Marcus Peinado, and Brent ByungHoon Kang. "Hacking in Darkness: Return-oriented Programming against Secure Enclaves." In: *26th USENIX Security Symposium (USENIX Security 17)*. 2017, pp. 523–539. url: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/lee-jaehyuk.

[100] Michael Schwarz, Samuel Weiser, and Daniel Gruss. "Practical Enclave Malware with Intel SGX." In: *Detection of Intrusions and Malware, and Vulnerability Assessment*. Cham: Springer International Publishing, 2019, pp. 177–196. doi: 10.1007/978-3-030-22038-9_9.

[101] Tyler Hunt, Zhiting Zhu, Yuanzhong Xu, Simon Peter, and Emmett Witchel. "Ryoan: A Distributed Sandbox for Untrusted Computation on Secret Data." In: *ACM Transactions on Computer Systems (TOCS)* 35.4 (Dec. 2018). issn: 0734-2071. doi: 10.1145/3231594.

[102] David Goltzsche, Manuel Nieke, Thomas Knauth, and Rüdiger Kapitza. "AccTEE: A WebAssembly-Based Two-Way Sandbox for Trusted Resource Accounting." In: *Proceedings of the 20th International Middleware Conference*. Middleware '19. 2019, pp. 123–135. doi: 10.1145/3361525.3361541.

[103] Samuel Weiser, Luca Mayr, Michael Schwarz, and Daniel Gruss. "SGXJail: Defeating Enclave Malware via Confinement." In: *22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019)*. 2019, pp. 353–366. isbn: 978-1-939133-07-6. url: https://www.usenix.org/conference/raid2019/presentation/weiser.

[104] Joongun Park, Seunghyo Kang, Sanghyeon Lee, Taehoon Kim, Jongse Park, Youngjin Kwon, and Jaehyuk Huh. *Stockade: Hardware Hardening for Distributed Trusted Sandboxes*. 2021. doi: 10.48550/ARXIV.2108.13922.

[105] Sergei Arnautov et al. "SCONE: Secure Linux Containers with Intel SGX." In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, 2016, pp. 689–703. url: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/arnautov.

[106] Le Guan, Peng Liu, Xinyu Xing, Xinyang Ge, Shengzhi Zhang, Meng Yu, and Trent Jaeger. "TrustShadow: Secure Execution of Unmodified Applications with ARM TrustZone." In: *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*. MobiSys '17. 2017, pp. 488–501. doi: 10.1145/3081333.3081349.

[107] Chia-che Tsai, Donald E. Porter, and Mona Vij. "Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX." In: *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. 2017, pp. 645–658. url: https://www.usenix.org/conference/atc17/technical-sessions/presentation/tsai.

[108] Andrew Baumann, Marcus Peinado, and Galen Hunt. "Shielding Applications from an Untrusted Cloud with Haven." In: *ACM Transactions on Computer Systems (TOCS)* 33.3 (2015). doi: 10.1145/2799647.

[109] Joshua Lind et al. "Glamdring: Automatic Application Partitioning for Intel SGX." In: *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. 2017, pp. 285–298. url: https://www.usenix.org/conference/atc17/technical-sessions/presentation/lind.

[110] Titouan Lazard, Johannes Götzfried, Tilo Müller, Gianni Santinelli, and Vincent Lefebvre. "TEEshift: Protecting Code Confidentiality by Selectively Shifting Functions into TEEs." In: *Proceedings of the 3rd Workshop on System Software for Trusted Execution*. SysTEX '18. 2018, pp. 14–19. doi: 10.1145/3268935.3268938.

[111] Brian McGillion, Tanel Dettenborn, Thomas Nyman, and N. Asokan. "Open-TEE – An Open Virtual Trusted Execution Environment." In: *2015 IEEE Trustcom/BigDataSE/ISPA*. Vol. 1. 2015, pp. 400–407. doi: 10.1109/Trustcom.2015.400.

[112] Microsoft. *Open Enclave SDK*. url: https://openenclave.io (visited on 2022-08-01).

[113] *Asylo*. url: https://asylo.dev/ (visited on 2022-10-01).

[114] Vasily A. Sartakov, Stefan Brenner, Sonia Ben Mokhtar, Sara Bouchenak, Gaël Thomas, and Rüdiger Kapitza. "EActors: Fast and Flexible Trusted Computing Using SGX." In: *Proceedings of the 19th International Middleware Conference*. Middleware '18. 2018, pp. 187–200. doi: 10.1145/3274808.3274823.

[115] "SGXKernel: A Library Operating System Optimized for Intel SGX." In: CF'17. 2017, pp. 35–44. doi: 10.1145/3075564.3075572.

[116] Meni Orenbach, Pavel Lifshits, Marina Minkin, and Mark Silberstein. "Eleos: ExitLess OS Services for SGX Enclaves." In: *Proceedings of the Twelfth European Conference on Computer Systems*. EuroSys '17. 2017, pp. 238–253. doi: 10.1145/3064176.3064219.

[117] Muhammad El-Hindi, Tobias Ziegler, Matthias Heinrich, Adrian Lutsch, Zheguang Zhao, and Carsten Binnig. "Benchmarking the Second Generation of Intel SGX Hardware." In: *Data Management on New Hardware*. DaMoN'22. 2022, pp. 1–8. doi: 10.1145/3533737.3535098.

[118] Intel. *3rd Gen Intel Xeon Scalable Processors Brief*. url: https://www.intel.com/content/www/us/en/products/docs/processors/xeon/3rd-gen-xeon-scalable-processors-brief.html (visited on 2022-10-02).

[119] Meysam Taassori, Ali Shafiee, and Rajeev Balasubramonian. "VAULT: Reducing Paging Overheads in SGX with Efficient Integrity Verification Structures." In: *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '18. 2018, pp. 665–678. doi: 10.1145/3173162.3177155.

[120] Sangho Lee, Ming-Wei Shih, Prasun Gera, Taesoo Kim, Hyesoon Kim, and Marcus Peinado. "Inferring Fine-grained Control Flow Inside SGX Enclaves with Branch Shadowing." In: *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, 2017, pp. 557–574. url: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/lee-sangho.

[121] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. "Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution." In: *27th USENIX Security Symposium (USENIX Security 18)*. 2018, pp. 991–1008. url: https://www.usenix.org/conference/usenixsecurity18/presentation/bulck.

[122] Stephan van Schaik, Marina Minkin, Andrew Kwong, Daniel Genkin, and Yuval Yarom. "CacheOut: Leaking Data on Intel CPUs via Cache Evictions." In: *2021 IEEE Symposium on Security and Privacy (SP)*. 2021, pp. 339–354. doi: 10.1109/SP40001.2021.00064.

[123] Stephan van Schaik, Andrew Kwong, Daniel Genkin, and Yuval Yarom. *SGAxe: How SGX Fails in Practice*. 2020. url: https://sgaxeattack.com/ (visited on 2022-10-13).

[124] Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, and Stefan Mangard. "Malware Guard Extension: Using SGX to Conceal Cache Attacks." In: *Detection of Intrusions and Malware, and Vulnerability Assessment*. 2017, pp. 3–24. doi: 10.1007/978-3-319-60876-1_1.

[125] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. "Software Grand Exposure: SGX Cache Attacks Are Practical." In: *11th USENIX Workshop on Offensive Technologies (WOOT 17)*. 2017. url: https://www.usenix.org/conference/woot17/workshop-program/presentation/brasser.

[126] Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. "Cache Attacks on Intel SGX." In: *Proceedings of the 10th European Workshop on Systems Security*. EuroSec'17. 2017. doi: 10.1145/3065913.3065915.

[127] Jo Van Bulck, Nico Weichbrodt, Rüdiger Kapitza, Frank Piessens, and Raoul Strackx. "Telling Your Secrets without Page Faults: Stealthy Page Table-Based Attacks on Enclaved Execution." In: *26th USENIX Security Symposium (USENIX Security 17)*. 2017, pp. 1041–1056. url: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/van-bulck.

[128] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. "Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems." In: *2015 IEEE Symposium on Security and Privacy*. 2015, pp. 640–656. doi: 10.1109/SP.2015.45.

[129] Daniel Gruss, Julian Lettner, Felix Schuster, Olya Ohrimenko, Istvan Haller, and Manuel Costa. "Strong and Efficient Cache Side-Channel Protection using Hardware Transactional Memory." In: *26th USENIX Security Symposium (USENIX Security 17)*. 2017, pp. 217–233. url: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/gruss.

[130] Ming-Wei Shih, Sangho Lee, Taesoo Kim, and Marcus Peinado. "T-SGX: Eradicating controlled-channel attacks against enclave programs." In: *Proceedings of the 2017 Annual Network and Distributed System Security Symposium (NDSS)*. 2017. doi: 10.14722/ndss.2017.23193.

[131] G. Chen, W. Wang, T. Chen, S. Chen, Y. Zhang, X. Wang, T. Lai, and D. Lin. "Racing in Hyperspace: Closing Hyper-Threading Side Channels on SGX with Contrived Data Races." In: *2018 IEEE Symposium on Security and Privacy (SP)*. 2018, pp. 178–194. doi: 10.1109/SP.2018.00024.

[132] Oleksii Oleksenko, Bohdan Trach, Robert Krahn, Mark Silberstein, and Christof Fetzer. "Varys: Protecting SGX Enclaves from Practical Side-Channel Attacks." In: *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. Boston, MA, 2018, pp. 227–240. url: https://www.usenix.org/conference/atc18/presentation/oleksenko.

[133] Sanchuan Chen, Xiaokuan Zhang, Michael K. Reiter, and Yinqian Zhang. "Detecting Privileged Side-Channel Attacks in Shielded Execution with Déjà Vu." In: *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. ASIA CCS '17. 2017, pp. 7–18. doi: 10.1145/3052973.3053007.

[134] Sajin Sasy, Sergey Gorbunov, and Christopher W. Fletcher. *ZeroTrace : Oblivious Memory Primitives from Intel SGX*. Published: Cryptology ePrint Archive, Paper 2017/549. 2017. doi: 10.14722/ndss.2018.23239.

[135] Saba Eskandarian and Matei Zaharia. "ObliDB: Oblivious Query Processing for Secure Databases." In: *Proc. VLDB Endow.* 13.2 (Oct. 2019), pp. 169–183. issn: 2150-8097. doi: 10.14778/3364324.3364331.

[136] Riad Ladjel, Nicolas Anciaux, Philippe Pucheral, and Guillaume Scerri. "Trustworthy Distributed Computations on Personal Data Using Trusted Execution Environments." In: *TrustCom 2019 - The 18th IEEE International Conference on Trust, Security and Privacy in Computing and Communications / BigDataSE 2019 - 13th IEEE International Conference on Big Data Sci-*

ence and Engineering. 2019. doi: 10.1109/TrustCom/BigDataSE.2019.00058. url: https://hal.archives-ouvertes.fr/hal-02269207.

[137] Julien Loudet, Iulian Sandu-Popa, and Luc Bouganim. "SEP2P: Secure and Efficient P2P Personal Data Processing." In: *EDBT 2019 - 22nd International Conference on Extending Database Technology*. 2019. doi: 10.5441/002/edbt.2019.14.

[138] Julien Mirval, Luc Bouganim, and Iulian Sandu Popa. "Practical Fully-Decentralized Secure Aggregation for Personal Data Management Systems." In: *33rd International Conference on Scientific and Statistical Database Management, SSDBM 2021*. SSDBM 2021: 33rd International Conference on Scientific and Statistical Database Management, Tampa, FL, USA, July 6-7, 2021. 2021, pp. 259–264. doi: 10.1145/3468791.3468821. url: https://hal.archives-ouvertes.fr/hal-03329878.

[139] Ludovic Javet, Nicolas Anciaux, Luc Bouganim, and Philippe Pucheral. "Edgelet Computing: Pushing Query Processing and Liability at the Extreme Edge of the Network." In: *CCGrid 2022*. Taormina, Italy, May 2022. doi: 10.1109/CCGrid54584.2022.00025. url: https://hal.inria.fr/hal-03666895.

[140] Luc Bouganim, Julien Loudet, and Iulian Sandu Popa. "Highly distributed and privacy-preserving queries on personal data management systems." In: *The VLDB Journal* (2022). doi: 10.1007/s00778-022-00753-1.

[141] Bohdan Trach, Alfred Krohmer, Sergei Arnautov, Franz Gregor, Pramod Bhatotia, and Christof Fetzer. "Slick: Secure middleboxes using shielded execution." In: *arXiv* (2017). doi: 10.48550/arXiv.1709.04226.

[142] Klaudia Krawiecka, Arseny Kurnikov, Andrew Paverd, Mohammad Mannan, and N. Asokan. "SafeKeeper: Protecting Web Passwords Using Trusted Execution Environments." In: *Proceedings of the 2018 World Wide Web Conference*. WWW '18. International World Wide Web Conferences Steering Committee, 2018, pp. 349–358. doi: 10.1145/3178876.3186101.

[143] Andreas Fitzek, Florian Achleitner, Johannes Winter, and Daniel Hein. "The ANDIX research OS — ARM TrustZone meets industrial control systems security." In: *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*. 2015, pp. 88–93. doi: 10.1109/INDIN.2015.7281715.

[144] Fan Mo, Ali Shahin Shamsabadi, Kleomenis Katevas, Soteris Demetriou, Ilias Leontiadis, Andrea Cavallaro, and Hamed Haddadi. "DarkneTZ: Towards Model Privacy at the Edge Using Trusted Execution Environments." In: *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*. MobiSys '20. 2020, pp. 161–174. doi: 10.1145/3386901.3388946.

[145] Fan Mo, Hamed Haddadi, Kleomenis Katevas, Eduard Marin, Diego Perino, and Nicolas Kourtellis. "PPFL: Privacy-Preserving Federated Learning with Trusted Execution Environments." In: *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*. MobiSys '21. 2021, pp. 94–108. doi: 10.1145/3458864.3466628.

[146] David Goltzsche, Tim Siebels, Lennard Golsch, and Rüdiger Kapitza. *Edgar: Offloading Function Execution to The Ultimate Edge*. Tech. rep. 2021. doi: 10.24355/dbbs.084-202111031112-0.

[147] Nicolas Anciaux, Luc Bouganim, Philippe Pucheral, Iulian Sandu Popa, and Guillaume Scerri. "Personal Database Security and Trusted Execution Environments: A Tutorial at the Crossroads." In: *Proc. VLDB Endow.* 12.12 (2019), pp. 1994–1997. doi: 10.14778/3352063.3352118. url: https://hal.inria.fr/hal-02269292/.

[148] Sumeet Bajaj and Radu Sion. "TrustedDB: A Trusted Hardware Based Database with Privacy and Data Confidentiality." In: *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*. SIGMOD '11. 2011, pp. 205–216. doi: 10.1145/1989323.1989346.

[149] Arvind Arasu, Spyros Blanas, Ken Eguro, Raghav Kaushik, Donald Kossmann, Ravi Ramamurthy, and Ramarathnam Venkatesan. "Orthogonal Security With Cipherbase." In: *6th Biennial Conference on Innovative Data Systems Research (CIDR'13)*. 2013. url: https://www.microsoft.com/en-us/research/publication/orthogonal-security-with-cipherbase/ (visited on 2022-10-13).

[150] Yongzhi Wang, Lingtong Liu, Cuicui Su, Jiawen Ma, Lei Wang, Yibo Yang, Yulong Shen, Guangxia Li, Tao Zhang, and Xuewen Dong. "CryptSQLite: Protecting Data Confidentiality of SQLite with Intel SGX." In: *2017 International Conference on Networking and Network Applications (NaNA)*. 2017, pp. 303–308. doi: 10.1109/NaNA.2017.48.

[151] C. Priebe, K. Vaswani, and M. Costa. "EnclaveDB: A Secure Database Using SGX." In: *2018 IEEE Symposium on Security and Privacy (SP)*. May 2018, pp. 264–278. doi: 10.1109/SP.2018.00025.

[152] Dhinakaran Vinayagamurthy, Alexey Gribov, and Sergey Gorbunov. "StealthDB: a Scalable Encrypted Database with Full SQL Query Support." In: *Proc. Priv. Enhancing Technol.* 2019.3 (2019), pp. 370–388. doi: 10.2478/popets-2019-0052.

[153] Wenchao Zhou, Yifan Cai, Yanqing Peng, Sheng Wang, Ke Ma, and Feifei Li. "VeriDB: An SGX-Based Verifiable Database." In: *Proceedings of the 2021 International Conference on Management of Data*. SIGMOD-/PODS '21. 2021, pp. 2182–2194. doi: 10.1145/3448016.3457308.

[154] Microsoft. *Azure SQL - Always encrypted with secure enclaves*. 2019. url: https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/always-encrypted-enclaves (visited on 2021-11-04).

[155] Edgeless Systems. *EdgelessDB*. 2021. url: https://www.edgeless.systems/products/edgelessdb/ (visited on 2021-11-02).

[156] Benny Fuhry, Raad Bahmani, Ferdinand Brasser, Florian Hahn, Florian Kerschbaum, and Ahmad-Reza Sadeghi. "HardIDX: Practical and secure index with SGX in a malicious environment." In: 26.5 (2018), pp. 677–706. doi: 10.3233/JCS-171103.

[157] Pratyush Mishra, Rishabh Poddar, Jerry Chen, Alessandro Chiesa, and Raluca Ada Popa. "Oblix: An Efficient Oblivious Search Index." In: *2018 IEEE Symposium on Security and Privacy (SP)*. 2018, pp. 279–296. doi: 10.1109/SP.2018.00045.

[158] Harshavardhan Unnibhavi, David Cerdeira, Antonio Barbalace, Nuno Santos, and Pramod Bhatotia. "Secure and Policy-Compliant Query Processing on Heterogeneous Computational Storage Architectures." In: *ACM SIGMOD/PODS International Conference on Management of Data 2022*. 2022. doi: 10.1145/3514221.3517913.

[159] Stephen JP Todd. "The Peterlee Relational Test Vehicle-a system overview." In: *IBM Systems Journal* 15.4 (1976), pp. 285–308. doi: 10.1147/sj.154.0285.

[160] Volker Linnemann, Klaus Küspert, Peter Dadam, Peter Pistor, R Erbe, Alfons Kemper, Norbert Südkamp, Georg Walch, and Mechtild Wallrath. "Design and Implementation of an Extensible Database Management System Supporting User Defined Data Types and Functions." In: *VLDB*. 1988, pp. 294–305.

[161] MySQL. *Adding Functions to MySQL*. url: https://dev.mysql.com/doc/extending-mysql/8.0/en/adding-functions.html (visited on 2022-10-08).

[162] Oracle. *User-Defined Functions*. url: https://docs.oracle.com/cd/B19306_01/server.102/b14200/functions231.htm (visited on 2022-10-08).

[163] Microsoft. *User-defined functions*. url: https://learn.microsoft.com/en-us/sql/relational-databases/user-defined-functions/user-defined-function (visited on 2022-10-08).

[164] Michael Godfrey, Tobias Mayr, Praveen Seshadri, and Thorsten von Eicken. "Secure and Portable Database Extensibility." In: *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*. SIGMOD '98. 1998, pp. 390–401. doi: 10.1145/276304.276339.

[165] Nicholas C. Wanninger, Joshua J. Bowden, Kirtankumar Shetty, Ayush Garg, and Kyle C. Hale. "Isolating Functions at the Hardware Limit with Virtines." In: *Proceedings of the Seventeenth European Conference on Computer Systems*. EuroSys '22. 2022, pp. 644–662. doi: 10.1145/3492321.3519553.

[166] Tobias Mayr and Praveen Seshadri. "Client-Site Query Extensions." In: *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*. SIGMOD '99. 1999, pp. 347–358. doi: 10.1145/304182.304213.

[167] Snowflake. *Secure UDF*. 2019. url: https://docs.snowflake.com/en/sql-reference/udf-secure.html (visited on 2021-11-04).

[168] Google. *Google BigQuery - Authorized Functions*. 2011. url: https://cloud.google.com/bigquery/docs/authorized-functions (visited on 2021-11-04).

[169] Stefan Brenner and Rüdiger Kapitza. "Trust More, Serverless." In: *Proceedings of the 12th ACM International Conference on Systems and Storage*. SYSTOR '19. 2019, pp. 33–43. doi: 10.1145/3319647.3325825.

[170] Bohdan Trach, Oleksii Oleksenko, Franz Gregor, Pramod Bhatotia, and Christof Fetzer. "Clemmys: Towards Secure Remote Execution in FaaS." In: *Proceedings of the 12th ACM International Conference on Systems and Storage*. SYSTOR '19. New York, NY, USA, 2019, pp. 44–54. doi: 10.1145/3319647.3325835.

[171] Ronald L Rivest, Len Adleman, Michael L Dertouzos, et al. "On data banks and privacy homomorphisms." In: *Foundations of secure computation* 4.11 (1978), pp. 169–180.

[172] R. L. Rivest, A. Shamir, and L. Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems." In: *Commun. ACM* 21.2 (1978), pp. 120–126. doi: 10.1145/359340.359342.

[173] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. "Evaluating 2-DNF formulas on ciphertexts." In: *Theory of cryptography conference*. Springer, 2005, pp. 325–341. doi: 10.1007/978-3-540-30576-7_18.

[174] Craig Gentry. "A Fully Homomorphic Encryption Scheme." PhD thesis. 2009. isbn: 9781109444506.

[175] Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. "CryptDB: Protecting Confidentiality with Encrypted Query Processing." In: *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. SOSP '11. 2011, pp. 85–100. doi: 10.1145/2043556.2043566.

[176] Paulo Martins, Leonel Sousa, and Artur Mariano. "A Survey on Fully Homomorphic Encryption: An Engineering Perspective." In: *ACM Comput. Surv.* 50.6 (2017). doi: 10.1145/3124441.

[177] Latanya Sweeney. "k-anonymity: A model for protecting privacy." In: *International journal of uncertainty, fuzziness and knowledge-based systems* 10.05 (2002). Publisher: World Scientific, pp. 557–570. doi: 10.1142/S0218488502001648.

[178] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. "L-Diversity: Privacy beyond k-Anonymity." In: *ACM Trans. Knowl. Discov. Data* 1.1 (2007). doi: 10.1145/1217299.1217302.

[179] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. "t-Closeness: Privacy Beyond k-Anonymity and l-Diversity." In: *2007 IEEE 23rd International Conference on Data Engineering*. 2007, pp. 106–115. doi: 10.1109/ICDE.2007.367856.

[180] Cynthia Dwork. "Differential Privacy." In: *Automata, Languages and Programming*. Ed. by Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener. Berlin, Heidelberg, 2006, pp. 1–12. doi: 10.1007/11787006_1.

[181] Shiva Prasad Kasiviswanathan, Homin K. Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. "What Can We Learn Privately?" In: *SIAM Journal on Computing* 40.3 (2011), pp. 793–826. doi: 10.1137/090756090.

[182] Stanley L. Warner. "Randomized Response: A Survey Technique for Eliminating Evasive Answer Bias." In: *Journal of the American Statistical Association* 60.309 (1965), pp. 63–69. doi: 10.1080/01621459.1965.10480775.

[183] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. "RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response." In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. CCS '14. 2014, pp. 1054–1067. doi: 10.1145/2660267.2660348.

[184] Bolin Ding, Janardhan Kulkarni, and Sergey Yekhanin. "Collecting Telemetry Data Privately." In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc., 2017. url: https://proceedings.neurips.cc/paper/2017/file/253614bbac999b38b5b60cae531c4969-Paper.pdf.

[185] A. Sabelfeld and A.C. Myers. "Language-based information-flow security." In: *IEEE Journal on Selected Areas in Communications* 21.1 (2003), pp. 5–19. doi: 10.1109/JSAC.2002.806121.

[186] Andrew C. Myers. "JFlow: Practical Mostly-Static Information Flow Control." In: *Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL '99. 1999, pp. 228–241. doi: 10.1145/292540.292561.

[187] Ruide Zhang, Ning Zhang, Assad Moini, Wenjing Lou, and Y. Thomas Hou. "PrivacyScope: Automatic Analysis of Private Data Leakage in TEE-Protected Applications." In: *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*. 2020, pp. 34–44. doi: 10.1109/ICDCS47774.2020.00013.

[188] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. "TaintDroid: An Information-Flow Tracking System for Real-time Privacy Monitoring on Smartphones." In: *ACM Trans. Comput. Syst.* 32.2 (2014). doi: 10.1145/2619091.

[189] Mingshen Sun, Tao Wei, and John C.S. Lui. "TaintART: A Practical Multi-Level Information-Flow Tracking System for Android RunTime." In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. CCS '16. 2016, pp. 331–342. doi: 10.1145/2976749.2978343.

[190] E. Novak, P. T. Aung, and T. Do. "VPN+ Towards Detection and Remediation of Information Leakage on Smartphones." In: *2020 21st IEEE International Conference on Mobile Data Management (MDM)*. 2020, pp. 39–48. doi: 10.1109/MDM48529.2020.00025.

[191] Jingjing Ren, Ashwin Rao, Martina Lindorfer, Arnaud Legout, and David Choffnes. "ReCon: Revealing and Controlling PII Leaks in Mobile Network Traffic." In: *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*. MobiSys '16. 2016, pp. 361–374. doi: 10.1145/2906388.2906392.

[192] Nabil R Adam and John C Wortmann. "Security-Control Methods for Statistical Databases: A Comparative Study." In: *ACM Computing Surveys* 21.4 (1989), p. 42. doi: 10.1145/76894.76895.

[193] Denning and Schlorer. "Inference Controls for Statistical Databases." In: *Computer* 16.7 (1983), pp. 69–82. doi: 10.1109/MC.1983.1654444.

[194] A. D. Friedman and L. J. Hoffman. "Towards a Fail-Safe Approach to Secure Databases." In: *1980 IEEE Symposium on Security and Privacy*. IEEE, 1980, pp. 18–18. doi: 10.1109/SP.1980.10018.

[195] J. F. Traub, Y. Yemini, and H. Woźniakowski. "The statistical security of a statistical database." In: *ACM Transactions on Database Systems* 9.4 (1984), pp. 672–679. doi: 10.1145/1994.383392.

[196] Dorothy E Denning. "Secure Statistical Databases with Random Sample Queries." In: *ACM Transactions on Database Systems* 5.3 (1980), p. 25. doi: 10.1145/320613.320616.

[197] Stefan Brenner, Michael Behlendorf, and Rüdiger Kapitza. "Trusted Execution, and the Impact of Security on Performance." In: *Proceedings of the 3rd Workshop on System Software for Trusted Execution*. SysTEX '18. 2018, pp. 28–33. doi: 10.1145/3268935.3268943.

[198] Anders T. Gjerdrum, Robert Pettersen, Håvard D. Johansen, and Dag Johansen. "Performance of Trusted Computing in Cloud Infrastructures with Intel SGX." In: *Proceedings of the 7th International Conference on Cloud Computing and Services Science, CLOSER*. 2017, pp. 668–675. doi: 10.1007/978-3-319-94959-8_1.

[199] Georges Hebrail and Alice Berard. *Individual household electric power consumption Data Set*. 2012. url: https://archive.ics.uci.edu/ml/datasets/individual+household+electric+power+consumption (visited on 2022-02-16).

[200] Microsoft Research Asia. *GeoLife GPS Trajectories*. 2016. url: https://www.microsoft.com/en-us/download/details.aspx?id=52367 (visited on 2022-02-16).

[201] Intel. *Intel SGX for Linux OS v2.15 - Developer Reference*. Sept. 2021. url: https://download.01.org/intel-sgx/sgx-linux/2.15/docs/Intel_SGX_Developer_Reference_Linux_2.15_Open_Source.pdf (visited on 2021-11-08).

[202] nPerf. *Baromètre des connexions Internet fixes en France métropolitaine*. 2020. url: https://perma.cc/DP8V-5ABT.

[203] Richard D Hipp. *SQLite*. url: https://www.sqlite.org (visited on 2022-08-03).

[204] TrustedFirmware. *Mbed TLS*. url: https://tls.mbed.org/ (visited on 2022-08-01).

[205] Thomas Knauth, Michael Steiner, Somnath Chakrabarti, Li Lei, Cedric Xing, and Mona Vij. *Integrating Remote Attestation with Transport Layer Security*. doi: 10.48550/ARXIV.1801.05863.

[206] Google. *FlatBuffers*. url: https://google.github.io/flatbuffers/ (visited on 2022-08-03).

[207] The ZeroMQ project. *ZeroMQ, An open-source universal messaging library*. url: https://github.com/zeromq/zmqpp (visited on 2022-08-07).

[208] Wt contributors. *Wt, C++ Web Toolkit*. url: https://github.com/emweb/wt (visited on 2022-08-07).