An Extensive and Secure Personal Data Management System Using SGX

Robin Carpentier
Univ. Versailles St-Q.-en-Yvelines
robin.carpentier@uvsq.fr

Floris Thiant
Inria Saclay
floris.thiant@inria.fr

Iulian Sandu Popa
Univ. Versailles St-Q.-en-Yvelines
iulian.sandu-popa@uvsq.fr

Nicolas Anciaux
Inria Saclay
nicolas.anciaux@inria.fr

Luc Bouganim
Inria Saclay
luc.bouganim@inria.fr

ABSTRACT

Personal Data Management System (PDMS) solutions are currently flourishing, spurred by new privacy regulations such as GDPR and new legal concepts like data altruism. PDMSs aim to empower individuals by providing appropriate tools to collect and manage their personal data and share computed results with third parties, thus requiring (i) a secure platform protecting the user's privacy and delivering strong guarantees on the outputs of user's data processing, and (ii) an extensible solution that supports all types of data-driven computations. In previous works, we analyzed these requirements and proposed an Extensive and Secure PDMS (ES-PDMS) logical architecture. This demonstration presents the first ES-PDMS prototype based on SGX enclaves, focusing on its security properties with the help of several concrete scenarios and interactive games.

1 INTRODUCTION

Over the past decade, successive steps have been taken to empower individuals with new legal means, from smart disclosure initiatives in the US to the right to data portability [5] and the new notion of data altruism enacted in the EU [4]. The ultimate goal is to enable individuals to collect and share their personal information and unlock innovative uses enabled by cross-exploiting multiple personal data sources. The technical corollary is the emergence of Personal Data Management Systems (PDMS) for individuals, giving rise to PDMS products such as Digi.me, Cozy Cloud, Personal Infomediaries [9], Solid/PODS [13], as well as large initiatives like Mydata.org, supported by data protection agencies.

Our in-depth analysis in [1] shows that the new PDMS paradigm, which causes a shift of data and execution to the PDMS owner's side (the terms user or PDMS owner are used interchangeably hereafter), introduces specific –and conflicting– requirements for extensiveness and security in data-oriented computations. To exemplify this trend, we consider the *Green bonus* scenario described below:

> ***'Green bonus' scenario:*** *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. Figure 1 proposes a possible, user centered, workflow: GPS traces are collected from a reliable service (e.g., Google Maps), then processed locally (e.g., by the user's PDMS) and the result is delivered to the employer with a proof of compliance.*

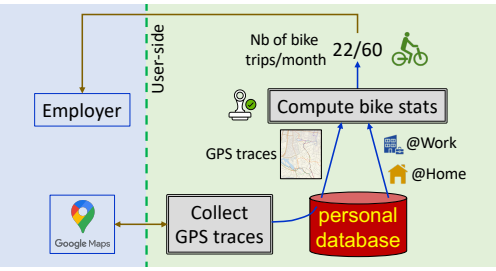

Figure 1: PDMS scenario example: 'Green bonus'.

One specificity in this scenario is to call for mutual security between stakeholders (employees, i.e., PDMS users, and employers, i.e., PDMS queriers): (i) *privacy* is needed to ensure employees that their detailed location history is not disclosed to third parties (including the employer); and (ii) *verifiability* must ensure employers that the computed number of trips is faithful (e.g., a reliable cloud service has provided the employee's location history and a specific processing code was used to aggregate the data). Finally, (iii) *extensiveness* is required, meaning that arbitrary –application specific– function code must be supported to collect data (e.g., scrapper) and compute the results (e.g., computation goes beyond simple SQL and requires an extension of the PDMS). Several similar scenarios are realistic, e.g., patients providing statistics to hospitals, privacy preserving power-meters, pay-as-you-drive, etc.

Traditional solutions resorting to external services to perform the data collection and/or computation would successfully ensure extensiveness and verifiability, but sharing personal data or credentials raise privacy concerns and a perceived risk of mass surveillance. On the other hand, existing PDMS products focus on the extensiveness aspect with many data collectors and applications, but fall short in providing privacy and verifiability as detailed in [1].

The recent democratization of Trusted Execution Environments (TEEs) at the user-side (e.g., Intel SGX [6] or ARM TrustZone) opens new perspectives for secure computation in the PDMS context [2]. A TEE provides hardware security based on trusted enclaves, separated from the operating system and untrusted applications [6]. An enclave provides three properties: (i) *isolation* ensures that the enclave code cannot be influenced by an attacker; (ii) *confidentiality* ensures that the enclave data is hidden from any other process by encrypting the RAM; (iii) with *attestation*, the enclave can prove that its results were indeed produced by its genuine code.

Several works have focused on new security models where trust lies on the hardware rather than on the user environment. For example, EnclaveDB [14], a TEE-based DBMS, could be a building block for a PDMS ensuring privacy and verifiability

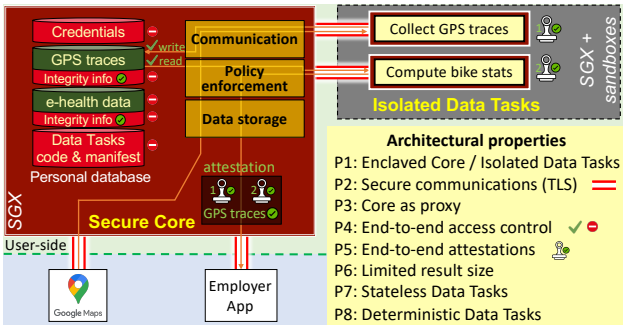

Figure 2: PDMS architecture and properties.

(adding appropriate access control and resorting to the TEE attestation capabilities [6]), but would fail to achieve extensibility. Indeed, in the PDMS context the -potentially untrusted- computations "move" to the data at the user-side. A TEE-based PDMS cannot simply be extended with traditional user-defined functions (UDF), since UDFs are assumed to be checked by admins, and thus, trusted, a too strong assumption in the PDMS context. Other TEE-based DBMS architectures, like stealthDB [8], favor extensibility by running large parts of the DBMS code outside enclaves, but this implies an honest-but-curious attacker model, considered unrealistic for user-side solutions.

Recent work on protecting query evaluation against side-channels attacks on TEEs, e.g., oblivious data structures [15] or differentially private techniques [12], is relevant to our proposal but is considered orthogonal. Finally, sandboxing techniques within enclaves (e.g., Ryoan [10], SGXJail [17]) protect the leakage of confidential data manipulated by untrusted enclaved code by restricting its address space. However, to guarantee the global security of the architecture combining trusted and untrusted code, all the untrusted code must be totally separated from the trusted computing base.

In this demonstration, we present our PDMS platform dealing with the conflicting objectives of extensiveness and security in the specific PDMS context, i.e., an ES-PDMS. Our platform is based on the logical architecture we have proposed in [1] and represents the first implementation of an ES-PDMS using SGX enclaves. The originality of our approach is to achieve extensibility through a set of isolated data-oriented tasks potentially untrusted by the PDMS owner, running alongside a trusted module which controls the complete workflow and limits data leakage [3]. The objective of the demonstration is twofold[1]: (1) to enable interactive visualization of the ES-PDMS secure workflow in different real-world scenarios to show the importance of the proposed security properties; (2) to demonstrate information leakage reduction in three attack scenarios through interactive games with the audience. Before presenting our interactive demonstration scenario in Section 3, we describe in Section 2 the architectural properties introduced to enforce the PDMS security and minimize data leakage.

2 EXTENSIVE AND SECURE PDMS

We introduce the architecture proposed in [1], present the security properties allowing to reconcile extensiveness and security and to minimize data leakage [3] and give some implementation details.

[1]Both objectives are illustrated in a video available at https://project.inria.fr/espdms/

2.1 Architecture Overview

Designing an extensive and secure PDMS architecture preserving usage with the guarantees defined above is a significant challenge, given the fundamental tension between security (small trusted code base) and extensiveness (rich data-driven tasks). In particular, the code extensions required to support advanced -specific- data-oriented computations cannot be part of a trusted code base for the PDMS. To solve this problem, we proposed in [1] a three-layer logical architecture where *Applications* (Apps), on which no security assumptions are made, communicate with a minimal *Secure Core* (Core) implementing basic operations on personal data, extended with *Isolated Data Tasks* (Data tasks), as shown in Figure 2.

Secure Core. The Core is a secure subsystem that is a Trusted Computing Base (TCB) ideally minimal, inextensible –potentially proven correct through formal methods– and isolated from the rest of the system. The Core provides all basic operations required to enforce the confidentiality, integrity and resiliency of the personal data hosted by the PDMS and hence is trusted both by the PDMS owner and by third parties that may access PDMS data. It must be the unique entry point to manipulate this data. The Core implements a data storage module, a policy enforcement module to regulate the data access performed by the other layers of the architecture, and a communication manager to securely communicate with other users, applications and third parties.

Isolated Data Tasks. Data Tasks are introduced as a means to support code extensions, needed to deal with application-specific personal data management. Data Tasks can run arbitrary application code, not necessarily trusted by the PDMS owner, thereby enabling extensiveness (like UDFs in traditional DBMSs). The idea is to control complex data-oriented tasks by (1) splitting their execution into Data Tasks evaluated in a sufficiently isolated environment to maintain control on the data accessed by the Core and delivered to the Apps in order to avoid any side effect in terms of data leaks, and (2) scheduling and verifying the execution of Data Tasks by the Core such that security and privacy can be globally enforced. Data processing pipelines may combine several Data Tasks and require some security properties. The PDMS must guarantee that the orchestration of these tasks cannot be tampered with without the caller being able to detect it.

Untrusted Apps. Any developer should be able to develop an application to ensure a wide and diverse application panel. However, the complexity of these applications (large code base, extensible and not proven) and their execution environment (e.g., web browser) make them vulnerable. Therefore, no security assumption is made on applications, which manipulate only authorized data resulting from Data Tasks but have no privileges on the raw data.

2.2 Architectural 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 under the control of the Core at the user side. Each App leverages one or several Data Tasks to execute its computations, expressed as logical execution plans. The *App manifest* includes essential information about the App and in particular the execution plan and its security configuration (e.g., access control rules). It should be validated by *e.g.*, a regulatory agency or the App marketplace and approved by the PDMS owner at the install time.

The ES-PDMS architecture implements the following security properties that are transparent to the PDMS user, who is not expected to be a security expert. Properties 1 to 5 are generic security properties that must be properly orchestrated to strengthen the security of our architecture. Security properties 6 to 8 were designed for the specific ES-PDMS context and were introduced in [3] along with a preliminary performance evaluation showing moderate overheads (other works involving enclave database management engines [14, 16] also attest acceptable overheads of less than 40%). Section 3 presents execution scenarios exploiting these properties.

P1. Enclaved Core/Data Tasks. The Core and each Data Task run in *individual enclaves* protecting the data and code execution confidentiality from the untrusted execution environment [6]. Besides, the code of each Data Task is *sandboxed* [10] within its enclave to preclude any voluntary data leakage outside the enclave by a malicious Data Task. The Apps run in the untrusted execution environment (locally or remotely).

P2. Secure communications. All the communications are classically secured using TLS (see Section 2.4) to ensure authenticity, integrity and confidentiality. Compared with a typical client/server setting, the overhead is limited to the establishment of additional secure TLS channels between the Core and the Data Tasks (once established, the cost is similar to inter-process communications).

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 the Data Task never has direct access to the "outside world".

P4. End-to-end access control. The input/output of the Data Tasks are regulated by the Core which enforces the *access control* rules for each Task required by an App based on the policies defined in the App manifest. For instance, a Data Task for collecting GPS traces from a web service has only the privilege to insert the collected data in the specified table (but cannot access any other data). Similarly, a Data Task computing the number of trips between two locations will only be supplied by the Core with the necessary GPS traces and the two GPS positions. Finally, the App only receives the final computation result from the Core (i.e., the number of trips) without being able to access any other data. This way the PDMS provides an end-to-end access control at the App level.

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 (e.g., data is protected against unauthorized updates from the PDMS owner). Second, P1 and P3 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.

2.3 Limiting Leakage

Taken together, all the properties presented above enforce the PDMS security and, in particular, the data confidentiality, precluding all the illegitimate data leakages except through the results that are delivered by the Core to the Apps. This section discusses more advanced properties specific to our three-layer architecture, which allow to further reduce this potential data leakage channel.

P6. Limited result size. Since the only possible leakage channel is through the final results the Apps receive from the Core, a malicious Data Task may use it to leak private data, i.e., include it within the bits codifying the result. Hence, the larger the result size is, the greater the probability for an attacker to leak more private data. A first measure that we propose is to *limit the result size* to the minimum required value allowing a correct execution of the computations (i.e., $log_2(size(domain(result))$ bits). In our example, the final results indicate the number of commutes by bike during a month, thus the result size should be limited to 6 bits. The result size, indicated in the validated and approved App manifest, is enforced by the Core before delivering any final result.

Even with limited result size, a malicious task could still obtain an important amount of private data over many computations. A malicious Data task can exploit data persistence, keeping a "state" between successive executions to maximize leakage. For instance, a Data Task could select some GPS traces data from the input, store it and progressively leak it over several executions while retaining the past leaks to avoid leaking the same data twice.

P7. Stateless Data Tasks. To preclude such behavior, we rely on *stateless Data Tasks* –without negative impact on usage as database computations are evaluated independently–. The stateless nature is an extension of the Data Task confinement by preventing access to the PDMS database managed by the Core or to the secure storage. A stateless Data task is instantiated for the sole purpose of answering a specific function call, after which it must be terminated and its RAM wiped (destroying the Data Task's enclave on SGX).

P8. Deterministic Data Tasks. Without being able to maintain state, a malicious Data Task could still exploit randomness as the next best thing to increase overall leakage over multiple executions. For example, at each execution, the Data Task selects a random fragment of the input to produce a new leak with high probability –even if the same computation is run on the same input–. A final security element is thus to impose *deterministic Data Tasks*, i.e., producing the same result for the same function code run on the same input. The in-enclave sandbox [10, 17] can be leveraged to provide this property by preventing access to any source of randomness, e.g., syscalls to random APIs or timer/date. Virtual random APIs are provided to preserve legitimate uses, e.g., the need for sampling, and are "reproducible", e.g., random numbers use a seed based on the function code and input set.

2.4 ES-PDMS Prototype Implementation

The demo platform runs on a server with Intel Xeon E-2276G 6-core CPUs with 12 MB cache and SGX 1-FLC support, 64GB RAM. The machine runs Ubuntu Linux 18.04 with SGX DCAP 1.41. The code of the Core and Data Tasks are written using Open Enclave (OE) [7], an SDK for developing enclave applications in C/C++. OE provides support for Intel SGX as well as preview support for ARM TrustZone, thus aiming to generalize the development of enclave applications across TEEs. The Core communication module uses the mbed TLS plugin integrated in OE to establish secure channels with the Data Task enclaves, remote parties or Apps. The Core data management module is based on SQLite (v3). The end-to-end attestation (P5) relies on the Intel SGX remote attestation with TLS[11] allowing to insert an attestation

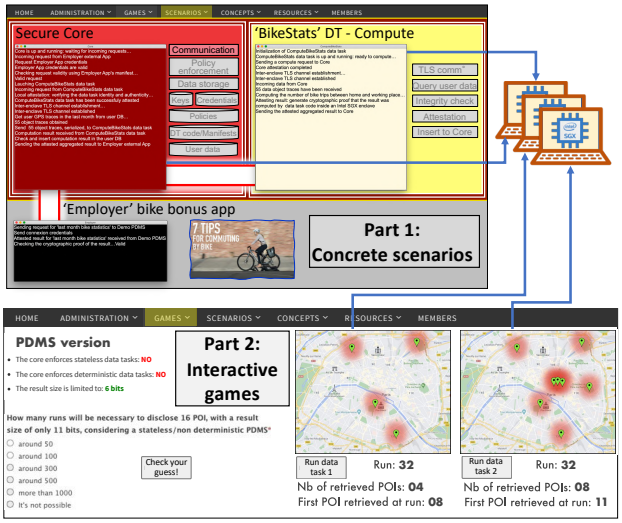

Figure 3: Demonstration interface of our PDMS prototype.

evidence based on the enclave identity (MRENCLAVE) and its signing identity (MRSIGNER) into a standard TLS certificate.

3 DEMONSTRATION SCENARIO

The scenario consists of two interactive parts [2] focusing on the execution workflow and mitigation of data leakage. In a first part, the attendee can run typical scenarios on PDMS instances to understand its behavior and can visualize execution traces highlighting the effect of properties P1 to P5 (see Section 2.2). The second part focuses on properties P4 to P8 (see Section 2.3) and is organized as an interactive game with the attendees through a website.

Part 1: visualizing ES-PDMS secure workflow with concrete scenarios. A set of PDMS scenarios are proposed to the attendee to process data from various external sources, including *Green bonus* and other scenarios exploiting e-health data produced by a smartphone App and collected from the French public social security service 'Ameli'. The attendee selects a scenario by validating the corresponding manifest and launches the underlying execution flow. The Core and Data Task enclaves are configured to produce detailed sets of traces related to architectural properties. The interface dynamically displays these traces in three consoles (for the Core, the Data Task, and the App or external data provider) and graphically highlights the modules and database access in each component (Figure 3, upper part). For *Green bonus*, the secure execution flow is split in two main stages: data collection and data computation. Our PDMS demonstration instruments these two steps.

Regarding data collection, after an initialization stage 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.

Part 2: demonstrating data leak containment with interactive games. Three concrete attack scenarios are proposed to the attendees in the form of games. A first attack is conducted by a PDMS user –an employee– in the *Green bonus* case. He falsifies his GPS data to get the maximum number of bike trips hoping to increase the financial benefit obtained. Attendees run the attack on PDMS instances compiled with different properties enabled/disabled and guess the impact on final results.

Then, we consider two attacks conducted by PDMS queriers –who are employers in the *Green bonus* scenario–, 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 –the code being potentially produced by the querier–. We first consider a corrupted compute Task used by the attacker to leak small size sensitive information (such as a password), to demonstrate the importance of *Limiting result sizeo* (P6), and making Data Task *Stateless* (P7) and *Deterministic* (P8). Then, we concentrate on more advanced Data tasks used in the *Green bonus* scenario, where attendees running different instances of PDMS have to guess which properties –among which P7 and P8– were enabled/disabled for each instance, based on the results.

A final challenge for the attendees is to indicate the maximum number of executions needed to reveal a given amount of points of interests in the PDMS user's GPS history, by optimally exploiting such attacks.

[2] A PDMS running on an SGX server at Inria will be available during the conference at https://project.inria.fr/espdms/

REFERENCES

[1] N. Anciaux, P. Bonnet, L. Bouganim, B. Nguyen, P. Pucheral, I. Sandu Popa, and G. Scerri. 2019. Personal Data Management Systems: The security and functionality standpoint. *Inf. Syst.* 80 (2019), 13–35.

[2] N. Anciaux, L. Bouganim, P. Pucheral, I. Sandu Popa, and G. Scerri. 2019. Personal Database Security and Trusted Execution Environments: A Tutorial at the Crossroads. *Proc. VLDB Endow.* 12, 12 (2019), 1994–1997.

[3] R. Carpentier, I. Sandu Popa, and N. Anciaux. 2021. Poster: Reducing Data Leakage on Personal Data Management Systems. In *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE Computer Society, 716–718.

[4] EU Commission. 25 October 2020. Proposal for a Regulation on European data governance (Data Governance Act), COM/2020/767. [eur-lex].

[5] EU Commission. 27 April 2016. Regulation (EU) 2016/679 (General Data Protection Regulation). [eur-lex].

[6] V. Costan and S. Devadas. 2016. Intel SGX Explained. *IACR Cryptol. ePrint Arch.* 2016 (2016), 86. http://eprint.iacr.org/2016/086

[7] Open Enclave. Accessed Nov. 2021. SDK https://openenclave.io/sdk/.

[8] A. Gribov, D. Vinayagamurthy, and S. Gorbunov. 2019. StealthDB: a scalable encrypted database with full SQL query support. *Privacy En. Tech. (PET)*, 370–388.

[9] T. Hardjono, D.L. Shrier, and A. Pentland. 2019. *Trusted Data, revised and expanded edition: A New Framework for Identity and Data Sharing*. MIT Press.

[10] T. Hunt, Z. Zhu, Y. Xu, S. Peter, and E. Witchel. 2018. Ryoan: A distributed sandbox for untrusted computation on secret data. *ACM TOCS* 35, 4 (2018), 1–32.

[11] Intel. 2019. Integrating Intel SGX Remote Attestation with TLS [pdf].

[12] M. Xu, A. Papadimitriou, A. Haeberlen, and A. Feldman. 2019. *Hermetic: Privacy-preserving distributed analytics without (most) side channels.* Technical Report MS-CIS-19-01. University of Pennsylvania.

[13] E. Mansour, S. Sambra, A. V.and Hawke, M. Zereba, S. Capadisli, A. Ghanem, A. Aboulnaga, and T. Berners-Lee. 2016. A Demonstration of the Solid Platform for Social Web Applications. In *WWW*. 223–226.

[14] C. Priebe, K. Vaswani, and M. Costa. 2018. EnclaveDB: A secure database using SGX. In *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 264–278.

[15] K. Ren, Y. Guo, J. Li, X. Jia, C. Wang, Y. Zhou, S. Wang, N. Cao, and F. Li. 2020. Hybridx: New hybrid index for volume-hiding range queries in data outsourcing services. In *ICDCS*. IEEE, 23–33.

[16] N. Weichbrodt, P-L. Aublin, and R. Kapitza. 2018. sgx-perf: A performance analysis tool for intel sgx enclaves. In *19th Int. Middleware Conf.,*. 201–213.

[17] S. Weiser, L. Mayr, M. Schwarz, and D. Gruss. 2019. SGXJail: Defeating Enclave Malware via Confinement. In *RAID*. USENIX Association, 353–366.