HAL open science

Personal Data Management Systems: The security and functionality standpoint

Nicolas Anciaux, Philippe Bonnet, Luc Bouganim, Benjamin Nguyen, Philippe Pucheral, Iulian Sandu-Popa, Guillaume Scerri

To cite this version:

Nicolas Anciaux, Philippe Bonnet, Luc Bouganim, Benjamin Nguyen, Philippe Pucheral, et al.. Personal Data Management Systems: The security and functionality standpoint. Information Systems, 2019, 80, pp.13-35. 10.1016/j.is.2018.09.002 . hal-01898705

HAL Id: hal-01898705

https://hal.science/hal-01898705v1

Submitted on 3 Feb 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License

Personal Data Management Systems: the Security and Functionality Standpoint

Nicolas Anciaux[a,b], Philippe Bonnet[c], Luc Bouganim[a,b], Benjamin Nguyen[d], Philippe Pucheral[a,b], Iulian Sandu Popa[a,b], Guillaume Scerri[a,b]

[a]*INRIA Saclay-Ile-de-France, Université Paris-Saclay, 1 Rue H. d'Estienne d'Orves, 91120 Palaiseau, France*
[b] *University of Versailles Saint-Quentin-en-Yvelines, Université Paris-Saclay, 45 avenue des Etats-Unis, 78035, Versailles Cedex, France*
[c] *IT Univ. of Copenhagen, Copenhagen, Denmark*
[d] *INSA Centre Val de Loire, 88 bd Lahitolle, 18022 Bourges, France & Université d'Orléans*

Abstract

Riding the wave of smart disclosure initiatives and new privacy-protection regulations, the Personal Cloud paradigm is emerging through a myriad of solutions offered to users to let them gather and manage their whole digital life. On the bright side, this opens the way to novel value-added services when crossing multiple sources of data of a given person or crossing the data of multiple people. Yet this paradigm shift towards user empowerment raises fundamental questions with regards to the appropriateness of the functionalities and the data management and protection techniques which are offered by existing solutions to laymen users. These questions must be answered in order to limit the risk of seeing such solutions adopted only by a handful of users and thus leaving the Personal Cloud paradigm to become no more than one of the latest missed attempts to achieve a better regulation of the management of personal data. To this end, we review, compare and analyze personal cloud alternatives in terms of the functionalities they provide and the threat models they target. From this analysis, we derive a general set of functionality and security requirements that any Personal Data Management System (PDMS) should consider. We then identify the challenges of implementing such a PDMS and propose a preliminary design for an extensive and secure PDMS reference architecture satisfying the considered requirements. Finally, we discuss several important research challenges remaining to be addressed to achieve a mature PDMS ecosystem.

Keywords: secure personal cloud, trusted execution environments

1. Introduction

Behaviors, movements, social relationships and interests of individuals are now constantly recorded, evaluated and analyzed in real-time by a small number of data aggregator companies [24]. This concentration negatively impacts privacy preservation and self-determination of individuals as well as innovation and fair competition between companies. Smart disclosure initiatives (e.g., Blue and GreenButton in the US[1], Midata in the UK[2], MesInfos in France[3]) are rising worldwide, aiming to restore individuals' control over their data and improve fairness in personal data management practices. The smart disclosure principle allows individuals to freely retrieve their personal data through a simple click, in a computer readable format, from the companies and administrations hosting them. According to the US government, this is a means to "*help consumers make more informed choices; give them access to useful personal data; power new kinds of digital tools, products, and services for consumers; and promote efficiency, innovation, and economic growth*" [51]. This fundamental principle has been recently translated into law with the *right to data portability* of the European General Data Protection Regulation (GDPR) [44].

Not only does smart disclosure allow individuals to be aware of the information collected about them, it also holds the promise of new services of high social and societal interest [51]. Indeed, individuals can now gather their complete digital environment in a so-called *Personal Cloud* or Personal Information Management Systems [1], Personal Data Server [3] or Personal Data Store [27]. A Personal Cloud is not only composed of data from many (previously) isolated information silos (e.g. secondary copies of data issued by their bank, employer, supermarket,

[1] https://www.healthit.gov/topic/health-it-initiatives/blue-button
[2] https://www.gov.uk/government/news/the-midata-vision-of-consumer-empowerment
[3] http://mesinfos.fing.org/

hospital) but also of primary data (e.g. produced by quantified-self devices and smart meters, photos taken with their smartphone or documents stored on their PC). This unprecedented concentration of personal data opens the way for new value-added services when crossing multiple data of a given person (e.g., crossing medical data with eating patterns or bank statements with shopping history) or crossing the data of multiple people (e.g., conducting an epidemiological study), all this under the concerned individual's control. While this will certainly not stop data aggregator companies' current practices, the Personal Cloud introduces an alternative way to develop *fairer* personal data management services using richer personal data. Hence, smart disclosure and the related Personal Cloud concept have become the cornerstone of what is called today *user empowerment*.

However, we should be cautious of a potential boomerang effect of user empowerment : returning individual's their data without providing them with the appropriate environment to exercise their control over it. Several companies are now riding the Personal Cloud wave and the spectrum of proposals in the internet sphere is highly diverse. *Online personal cloud* solutions (e.g., CozyCloud, Digi.me, BitsAbout.Me to only cite a few) propose a centralized web hosting of personal data combined with a rich set of services to collect personal data from various sources, store them and cross-exploit them. This approach assumes, by construction, that individuals do not question the honesty of the hosting company (including the honesty of the employees) nor its capacity to defeat severe attacks, since centralization creates by essence a massive honeypot. *Zero-knowledge personal cloud* solutions (e.g., SpiderOak [26], Sync) mitigate this strong trust assumption by offering a fully encrypted (yet still centralized) data store, but impose a new responsibility (managing the encryption keys) on the individuals, thus trading user friendliness and rich services for improved security. *Home cloud software* solutions (e.g., OpenPDS [27], DataBox [23, 39]) build upon the paradigm of local/edge computing by considering that the raw personal data should remain stored physically close to the user. Hence, they advocate for a decentralized approach where individuals install local personal servers on their own equipment (e.g., PC). *Home cloud plugs* (e.g., CloudLocker, Helixee) go further in this direction by offering a dedicated box that can store TBs of data, run a server and simply be plugged on an individual's home internet gateway, alleviating the burden of installing and administering a server. Finally *tamper-resistant home clouds* are home cloud plugs integrating secure chips on their hardware board to improve their resistance to confidentiality attacks, viruses and ransomwares. All these alternatives belong to the large, fuzzy, personal cloud system family but neither provide the same set of functionalities nor consider the same threat model.

This diversity of solutions raises important questions. Which functionalities are really mandatory in the personal cloud context? Which threat model better captures the various uses and architectural environments of the personal cloud? Do solutions exist combining the required set of functionalities and appropriate threat model? If not, where does the difficulty stem from? Can solutions be devised by adapting existing corporate cloud-based techniques or is the personal cloud problem fundamentally different, thus imposing a deep rethink of these techniques? These questions are important for the data management research community and for the individual as well. By leaving these questions without answers, the risk is high to see the Personal Cloud paradigm be nothing but a missed attempt to reach a better regulation of the management of personal data, and maybe one of the last.

This paper precisely tries to answer these questions by making the following contributions:

- *Personal cloud solutions categorization*: we review, compare and categorize the various personal cloud alternatives sketched above in terms of provided functionalities and targeted threat model. The functionality axis is organized so as to cover all important steps of the personal data life cycle, from collection to storage, recovery, individual and collective exploitation of personal data. The threat model axis tries to encompass all actors contributing to the use of a personal cloud platform, from the personal cloud service provider, the applications providers, to the individual and its storage and computing environment. Beyond identifying the main expected features and privacy threats that need to be addressed, we also show that existing alternatives do not cover all of these features and threats, and cannot be combined for this purpose.
- *Definition of an Extensible and Secure Personal Data Management System (ES-PDMS)*: we propose a definition of what an *extensive* (combining all functionalities) and *secure* (circumventing all the threats) *Personal Data Management System* should be. Therefore, we analyze the specificities of each functionality in the light of the individual context considered in this paper, and we deduce the corresponding security properties to achieve them. This has not been done until now, as existing personal cloud solutions have mostly been derived from their corporate counterparts.
- *Definition of an ES-PDMS reference architecture*: we have the strong belief that many security issues are rooted in architectural choices. Thus, we propose an abstract design for an *ES-PDMS* reference architecture satisfying the properties we will have defined. We then illustrate how this abstract architecture can be instantiated in different

concrete settings. For the sake of generality, this design makes no assumption on the PDMS data management model itself (models and languages to define, manipulate and share objects entering in a PDMS).

- *Research issues related to PDMS architectures*: finally, we review a set of important research issues which remain to be investigated concerning the definition and security of PDMS architectures.

The rest of the paper follows this same structure, one section being devoted to each contribution, followed by a conclusion highlighting the expected impact of this work.

2. Existing personal cloud solutions

The Personal Cloud concept originally appeared under different names such as *Personal Information Management Systems* [1], *Personal Data Server* [3] or *Personal Data Store* [27]. It attracts today significant attention from both the research and industrial communities. This section provides a review of existing personal cloud solutions, representative of current approaches, to help understand the fundamental aspects in terms of functional requirements and security/privacy threats.

We distinguish cloud data management solutions designed for the corporate/enterprise context from those targeting individuals, i.e., tailored for personal use and referred to as Personal Clouds in this section. Corporate cloud solutions offer digital tools to employees including, e.g., file storage space, email/agenda applications, file sharing with other employees based on permissions, user management and authentication based on LDAP repositories. They come as enterprise cloud tools either implemented as-a-service in the cloud or hosted on a server owned by the company and managed by internal administrators, such as SeaFile, Pydio, ownCloud/NextCloud, Sandstorm or Tonido/FileCloud. In terms of data management, the primary foci are multi-user features, workspace and user management, authentication, access control, privilege settings, administration and analysis tools. Such solutions do not apply to the personal cloud case and hence are not further detailed in this state of the art. Showing to which extent and detailing the major differences between individual and corporate-oriented solutions, is precisely one of the goals of the paper.

In contrast, solutions tailored for personal (and generally private) use are mono-user and seek to help users manage their entire digital life, i.e., by providing connectors to external data sources (e.g., bank, hospital, employer, social network, etc.), by allowing cross-data computation usages (e.g., linking the bank records of the individual with corresponding bills and email confirmations) and community uses based on groups of users sharing data for a social benefit (e.g., epidemiological study in a community of patients), by permitting their installation and configuration by laymen, and by helping individuals (rather than IT experts or administrators) understand and control data dissemination.

In the rest of this section, we thus deliberately focus on solutions tailored for individuals. We first review the online personal cloud solutions resorting to a personal cloud provider or remote storage service, then present variants offering additional security guarantees including zero-knowledge data stores, and finally analyze some more decentralized 'home cloud' proposals where the data is stored user-side, using purely software-based solution, hardware plugs or tamper resistant devices. We conclude this overview with a summary of the main features and security/privacy threats considered by existing solutions. This state-of-the-art analysis provides the needed material to derive in the next section the main underlying data management functionalities and related security goals of any extensive (in terms of database functionalities) and secure PDMS.

2.1. Online Personal Cloud solutions

Many online personal cloud solutions flourish today such as CozyCloud, Digi.me, Meeco, BitsAbout.Me or Camilistore/Perkeep to name a few. Governmental programs like MyData.org in Finland, MesInfos.fing.org in France or MyDex.org in the UK, target the same objective. These initiatives provide online personal cloud solutions to help users gather and store all their personal data in the same place and in a usable format, with the possibility to cross-exploit it through various applications. In terms of privacy and security, a common claim of these solutions is to proscribe any secondary usage (and in particular monetization of personal data) by the personal cloud provider and to guarantee to their users that their personal data is never disclosed to third parties except on their explicit request. The main functionalities advertised and the corresponding privacy promises of such systems are further described below.

Data collectors. Most of the solutions mentioned above build on recent regulations like the GDPR (including the EU 'right to personal data portability') or smart disclosure initiative (e.g., Blue and GreenButton in US), and provide *data collectors* which can automatically feed a personal cloud with user's data originating from different online services. Data collectors act as data bridges, which retrieve personal data on the behalf of the user (i.e., using her credentials) from external data sources. Usually, data collectors rely on web scrapping technologies (e.g., Cheerio, Weboob) which act on the behalf of the user on a target website to collect their personal data. For example, CozyCloud (through the 'CozyCollect' application) and Digi.me provide their users with a catalog of connectors to retrieve many kinds of personal data, including financial data (e.g., from banks or PayPal), administrative data (e.g., electricity or telco bills and consumption traces, insurance contracts), social network data (e.g., from Facebook or Pinterest accounts), music (e.g., Spotify), medical information (e.g., from social security institutions, hospitals, or Blue Button compliant sites) or fitness data (e.g., Fitbit), to cite only a few. BitsAbout.Me also provides data collectors but focusses instead on the web activity trails of the user and targets personal information such as Google geolocation histories, Facebook 'likes' graphs and YouTube histories.

Cross-data computation services. Online personal cloud solutions allow integrating personal data usually scattered across distinct and closed data silos, which opens the way for novel applications able to cross-exploit it. Digi.me provides transversal data services, transversal data searches, and data sharing between different apps. In CozyCloud, the apps interact with the Cozy data system to access documents stored in a CouchDB engine, where each document is stored in a JSON format with an associated 'doctype' family (e.g., bank, photos, bills, etc.). The list of existing doctypes are defined and published by Cozy such that Cozy app developers can conform to a common scheme and easily identify the documents of interest (e.g., a finance app may be granted access to all the 'bank' and 'bills' doctypes and cross-exploit them to link each bill to a corresponding bank transaction record). In Meeco, personal data is organized within *life-tiles*, i.e., datasets defined by the user which gather specific personal information or files uploaded by the user (e.g., photos taken around Christmas in a life-tile entitled 'Christmas'), and *web-tiles*, i.e., user defined goals (e.g., purchase intents) or user's activities on specified websites (e.g., amazon.com).

Trusted data storage. The personal data associated with a given personal cloud user is stored online, but within a data store which belongs to a single user. In CozyCloud and Meeco, the user's personal cloud is hosted and secured by the cloud service provider on behalf of the personal cloud owner (e.g., using server-side encryption). For instance, Meeco has chosen a cloud server in Australia to comply with the strict local privacy regulations, while Cozy can be deployed using any cloud service provider. In some cases (e.g. Cozy), expert users can opt for a self-hosted instance, which is close to the home cloud approach considered in Section 2.2. In Digi.me, data is encrypted and stored where the user whishes (e.g., on Dropbox, Google Drive or MS OneDrive). Encryption keys are stored on a user's device and derived from a user password. Keys are unlocked at connection time and used server side by data collectors and personal applications and are only retained for the duration of the session. In BitsAbout.Me, a *Personal Data Store* is dedicated to each user and is hosted encrypted in a datacenter in the EU or Switzerland (chosen for the high level of legal privacy protection offered by the GDPR and Swiss laws). Camilistore/Perkeep has a rather different approach as its goal is to provide a personal storage for life (above 100 years) to individuals. The solution thus focuses on providing users with easy means to generate a (searchable) personal data archive (storing all their, e.g., Tweets, social media photos, etc.) on a personal (device or cloud) store, independently of the websites hosting the data (e.g., Twitter, Instagram, etc.).

Trust model. All the above-mentioned solutions make strong privacy promises to the users in order to gain their trust. In particular, the personal cloud provider commits to never observe nor exploit the users' personal data for *secondary usages* not advertised to the personal cloud owner such as data monetization. Although the different proposals vary widely in the way the personal cloud provider effectively tries to gain the users' trust, it mainly relies on three arguments. The first argument is linked to the trust users may put in the *security standards* of authentication, communications and data encryption. For instance, Digi.me authenticates applications, encrypts communication channels using SSL and encrypts passwords with RSA 2048-bit (FIPS compliant). CozyCloud also follows the current security best practices in terms of users' passwords, connections and data at rest encryption. BitsAbout.Me declares that the decryption keys are expunged from its servers at user's disconnection and therefore the server can only access the data during the time of a session. The second argument is related to the *virtuous legal and economic frameworks* to which the cloud provider is bound. Typically, most providers underline a high degree of independence enacted by contract between the storage provider and the data owner. As explained above, Meeco or BitsAbout.Me argue that the location of their servers is chosen for the high level of legal privacy protection, and Digi.me lets users select the cloud storage of their choice. In some cases, users can

choose to have their personal cloud instance hosted directly by the cloud provider (e.g., CozyCloud) or by any other trusted third party of their choice (e.g., OVH, Dropbox, etc.). CozyCloud follows a similar principle, following the motto that 'users will stay because they can leave'. In addition, such personal cloud providers claim to adopt an economic model creating a virtuous circle for privacy and promise not to monetize the personal data provided by the users (or unless they explicitly demand it). The third argument to gain users' confidence is linked to the *transparency or auditability* of the code of the applications and the personal cloud platform. For example, CozyCloud's applications and data system code are open source. The main consequence is that expert users can review the code or audit it such that any black box effect is avoided.

Overall, these solutions focus on a very similar set of functionalities, which cover the collection of personal data, storage in an individual personal data store, and the integration of the data such that transversal information processing is made possible. However, while most initiatives claim to guarantee users' privacy, these approaches mainly rely on legal and economic frameworks with an unclear impact on the technical means to enforce security and privacy guarantees. Moreover, these approaches implicitly rely on very strong hypotheses in terms of security: (i) the personal cloud provider, employees and administrators are assumed to be *fully-honest*, and (ii) the overall personal cloud code as well as the whole set of personal applications and services running on top of it are considered *trusted*. Common security and privacy threats remain thus insufficiently addressed. Typically, data leakage resulting from attacks conducted against the personal cloud provider or the applications (which could be granted access to large subsets of raw personal data), or resulting from human errors, negligence or corruption of personal cloud employees and application developers, cannot be avoided in practice. This is critical because such solutions rely on a centralized cloud infrastructure settings which exacerbate the risk of exposing a large number of personal cloud owners, and hence may be subject to many sophisticated attacks.

2.2. Zero-knowledge-based Personal Clouds

Zero-knowledge personal clouds such as SpiderOak or Sync and to a certain extent MyDex or Digi.me mentioned above, propose architectural variations of the online Personal Cloud solutions in particular to mitigate some of the internal privacy issues raised by the strong assumption that the service provider is trusted. These solutions focus in particular on ***secure storage*** and ***backup***. These functionalities, as well as elements of the corresponding privacy threats, are sketched below.

Secure storage. In most of the personal cloud solutions offering zero-knowledge storage, data is stored encrypted in the cloud and the user inherits the responsibility to store and manage the encryption keys elsewhere (and never transmit it to the outside nor to the personal cloud provider). In SpiderOak, the personal cloud provider knows the number of encrypted data blocks produced by a given user, but not their content nor the associated metadata information (e.g., folder or file names). The encryption key of a user is derived from the user's password (i.e., password-based encryption). Note that file deduplication (i.e., storing a file only once at server side if several users hold that file) is not feasible as it would result in increasing the knowledge of the server. Sync uses password-based encryption techniques as well, but focuses more on the synchronization issues between the different devices of a same user. MyDex also offers a zero-knowledge service following privacy-by-design principles, but the system is here separated in two parts: (i) a front-end service which is in charge of retrieving the private keys from the users at connection time, encrypting and decrypting the personal data of the client during the session, and expunging the keys at disconnection; and (ii) a back-end service which stores collections of encrypted files. Note that an implicit assumption here is that the two parts cannot collude, and that relying on a front-end service at server side (instead of a client-side implementation) to manage the cryptographic keys weakens the zero-knowledge claim.

Secure backup. A common asset advertised by many zero-knowledge personal cloud services is secure backup, as a means to recover personal data when faced with a ransomware attack, personal device failure or unexpected data deletion. Thus, most zero-knowledge solutions, like SpiderOak, propose point-in-time recovery, such that users can recover any previous version of their personal files at a given date in the past. Note, however, that the user must assume the responsibility of storing and managing the encryption keys, since managing them server side would conflict with the zero-knowledge nature of these solutions.

Trust model. The threats considered by these solutions include (i) an attacker who compromises the personal cloud provider (i.e., addresses the case of *data snooping* and *data leakage*), (ii) a personal cloud provider that would want to make non-advertised usages on the customers' data content (i.e., *secondary usages*, e.g., personal

data monetization), and (iii) a *client device failure* or corruption (e.g., *ransomware attack*). Note however that the term 'zero-knowledge' does not refer here to its cryptographic definition counterpart, since the personal data access patterns are not supposed to be hidden from the personal cloud provider. This consideration recently led to adopt the term 'no-knowledge' data store. A recent analysis of the threat model considered by SpiderOak (which applies to the zero-knowledge personal cloud providers as discussed here) is presented in [26]. In a nutshell, the threat model assumes *Honest-but-curious* or *Malicious* personal cloud providers, and a *trusted client application and device* (at least, considered trusted before the time of failure or before a ransomware acts).

In conclusion, compared to the basic online personal cloud (see Section 2.1), the zero-knowledge personal cloud solutions offer a higher level of security since the personal cloud provider cannot access personal data in clear. However, the price to pay is a *minimalist functionality*, i.e., the difficulty to develop advanced services on top of zero-knowledge personal clouds, which reduces the uses of a zero-knowledge personal cloud to those of a robust personal data safe. Moreover, since data processing (beyond basic storage) cannot be delegated to the server, data-oriented treatments are embedded into the client applications, shifting the security and privacy issues to the client's side. On the other hand, the assumption of an honest client application (which has access to the decryption keys and to the raw data in clear) on which such cloud solutions rely, may be too strong to hold in practice (due to, e.g., the ubiquity of viruses), thus creating a vicious circle.

2.3. Home cloud software

Other personal cloud initiatives, called 'home cloud' hereafter, build upon the paradigm of local/edge computing. These initiatives consider that raw personal data should remain stored at the extremities of the network (e.g., within the user's equipment or close to the IoT device which produced it) as a means to circumvent the intrinsic security risks of data centralization (i.e., corruption of the server resulting in massive data leakage and illicit data usages). In this Section, we first discuss purely software-based solutions, and then move on to the case of hardware-based proposals in the next two Sections.

Some remarkable representatives of home cloud software solutions are OpenPDS [27] and DataBox [23, 39]. Both focus mainly on new privacy models allowing users to reduce the amount of personal data exposed to remote parties (i.e., data services or other users) and audit data exchanges. The core of these proposals is based on a *trusted storage* hosted locally on the user's device or at the edge of the network, combined with *cross-computations* and *data sharing* such that users may consent revealing only query results to third parties instead of disclosing large amounts of sensitive raw data.

Trusted storage. OpenPDS is a personal cloud solution which allows a user to accumulate personal data about her (e.g., web or shopping preferences, location traces) on her device (e.g., smartphone) and which provides a privacy preserving framework to explore this data. In Databox, the raw data storage is based on several isolated local stores, each data store being associated with a given source of personal raw data (e.g., one store per IoT device or sensor equipping the user), all located at the edges of the network. In both cases, data storage is considered trusted because it is managed locally on a user device.

Cross-computations and data dissemination. For both functionalities, the goal is circumventing the problem of third-parties being granted access to large amounts of user's raw data. Data sharing in OpenPDS is based on a framework called *Safe Answer* [27], a query-answering system used to analyze (i.e., cross-compute) the personal data collected by the individual and minimize the information about the data exposed to third parties or applications. The idea is to answer precise questions rather than externalize the complete set of raw data on which the queries are processed. In the same vein, DataBox [23, 39] proposes techniques inspired by the Human-Data Interaction (HDI) paradigm to enable individuals to understand what data is collected about them and how it is processed. The proposal is based on separating the raw data stores from other stores dedicated to materialize aggregated query results, which can be made accessible to remote third parties. The proposed model relies on the ability to log all data accesses and data flows, and provides audit capabilities to simplify the users' control and understanding of the effective dissemination of their personal data.

Trust model. The focus of both OpenPDS and DataBox is not on security and enforcement, but on the study of the aforementioned functionalities under the angle of new privacy models and edge computing. An implicit trust assumption is that the (local/edge located) data stores hosting the personal raw data are *trusted* as well as the data system used to implement these functionalities (i.e., the cross-computation engine used to produce the aggregated results to answer queries, as well as the sharing model and the audit framework). The software architecture of

Databox advocates the use of Docker containers (MirageOS unikernel being mentioned as a long-term alternative) to isolate certain data computations on the raw data. However, formal security guarantees are not discussed and the proposals do not focus on solutions to ensure that the proposed privacy models cannot be bypassed.

Compared with zero-knowledge solutions, formal security guarantees (in particular on the backup service) are lost. But interestingly, the impact in terms of 'minimalist functionality' is alleviated since advanced data services could potentially be provided as part of the personal cloud platform (which is obviously not compatible with the zero-knowledge guarantee). In addition, such solutions target user privacy protection against over-privileged third-parties and applications under the strong security assumption of having a fully-secured personal cloud software user side.

2.4. Home cloud plugs

Home cloud plug solutions such as Lima [4], Helixee [5], CloudLocker[6] and MyCloud[7], distinguish themselves from the previous approaches by providing solutions helping the users to self-host their personal data at home on a dedicated hardware platform. These solutions take the form of hardware plugs that can store TBs of data, which are synchronized with all the devices of the owner and can be accessed online.

Trusted storage and backup. In Lima, the hardware plug is connected to the user's internet home-box and to an external disk drive. Other solutions, like Helixee, directly integrate the disk drive into the plug. Personal data is stored encrypted locally and is made accessible by the home cloud plug, which holds the encryption keys, to a set of personal devices authorized by the user (e.g., her smartphone and laptop). Usually, a central server can be used as DNS (as in Lima) to establish a remote connection between the users' devices and the hardware plug. The system can then be backed-up automatically using a second plug or a remote encrypted archive locked by the user password.

Trust model. The main privacy benefit of home cloud plugs comes from the absence of delegation to a central cloud server. In terms of security however, the implicit assumption is strong, as the home cloud hardware and software platform must be *trusted*. This hypothesis is supported by the fact that the hardware plug constitutes a rather closed (dedicated) platform since no application except the software managing the plug is supposed to run on it. This limits the attack surface compared with richer devices (such as a smartphone). However, no formal security guarantees are provided to the user concerning the self-hosted platform and the application services running on top. This can put the user's raw data at risk considering that unsecured end-user devices are accessing it. For instance, the DynDNS attack of November 2016, which infected unsecured end-user devices (e.g., printers, IP cameras and residential gateways) with a malware, illustrates the vulnerability of such home-based solutions.

To conclude, the focus in terms of personal data uses is again mainly on trusted storage and backup, and to a certain extent basic data sharing to support data synchronization between all the devices of the user. However, collaborative uses involving personal data from multiple individuals are, to the best of our knowledge, outside of the scope of these approaches. In terms of privacy and security, the gain compared to home cloud software solutions is, to some extent, a better controlled client execution environment for the personal cloud software, which nevertheless does not provide strong security guarantees. These two complementary approaches pose a problem in terms of safely extending the data related functionalities. Indeed, implementing a new advanced data service would require either to extend the trusted code hosted on the hardware plug, which should be considered as a closed platform for security reasons, or to add it as an external app, which in this case would run on vulnerable client devices.

2.4.1. Tamper-resistant home cloud

To improve the security of home cloud plugs, research proposals like Personal Data Server (PDS) [3] and Trusted Cells [8] introduce secure (i.e., tamper-resistant) hardware at the network edges to manage the user's personal data. These approaches propose to embed a minimal Trusted Computing Base (TCB) dedicated to data

[4] https://meetlima.com/tech.php?lang=en
[5] http://www.helixee.me/
[6] https://www.cloudlocker.eu/en/index.html
[7] https://mycloud.com/

management in the secure element of smart phones, set-top boxes or portable USB tokens to form a global decentralized secured data platform.

Secure storage. The PDS approach builds upon the tamper resistance of secure chips (e.g., smart cards, secure tokens). A DBMS engine is embedded in a secure chip, and hence inherits its security properties. The database metadata are stored in the internal memory of the chip are thus become tamper resistant, and the database itself (the data, indexes, logs, etc.) is cryptographically protected and stored in an external Flash memory (e.g., a raw NAND Flash chip or a MicroSD card linked by a bus to the microcontroller like in PlugDB[8]).

Secure cross-computations. Simple query evaluation and an access control engine can be integrated into the PDS engine running in the secure chip [9, 33]. The secure cross-computations are however limited to simple database queries.

Secure distributed computations. The possibilities of crossing data belonging to multiple individuals (e.g., performing statistical queries over personal data, computing queries on social graphs or organizing participatory data collection) while providing strong privacy guarantees have been explored in the context of a network of PDSs so that each user can keep control over her data. The personal data is stored locally in each user's PDS and the execution takes place on a hybrid infrastructure called an asymmetric architecture: on the one hand the PDSs of the participants are secure (i.e., behave honestly) but have low computation power, on the other hand, they are supported by an untrusted cloud infrastructure (e.g., honest-but-curious) implementing an IaaS or PaaS with significant storage and computing power. Different algorithms and computing paradigms have been studied on this architecture, from SQL aggregates [53] to special aggregation in a mobile participatory sensing context [52]. In all cases, the challenge is to trade privacy for performances depending on the equilibrium between the secure computations executed by the secure PDSs and the ones delegated to the untrusted cloud infrastructure.

Trust model. In this line of work, each PDS is assumed to be trusted. This trust assumption comes from several factors: (i) the PDS software inherits the tamper resistance of the hardware and can be certified according to the Common Criteria, making hardware and software attacks highly difficult, and (ii) the embedded database can be auto-administered due to its simplicity (in contrast with multi-user server counterparts) which precludes DBA attacks. However, a PDS cannot provide all the required database functionalities without resorting to an external infrastructure (e.g., distributed queries involving several PDSs as described above require external supporting servers). While the PDSs can be assumed to be fully trusted, the supporting communication and computation infrastructure is considered as the main adversary. It is considered as a *malicious adversary* having *weakly malicious intents* [17]. This means that it may deviate from the protocols it implements in order to infer personal information, but only tries to cheat when it cannot be detected by any PDS user. This assumption is commonly made for cloud services, as publicly advertising data leaks would cause important (financial) damages to the underlying service provider.

In conclusion, a high security level can be achieved since tamper resistant hardware is used. However, only rather simple queries can be addressed, and the underlying query engine being part of the TCB (running inside the secure microcontroller) must be proven secure. In addition, this approach leads to a non-extensible data system for two main reasons. First, supporting any new data and query model would require redesigning the underlying data storage, indexing and query processing techniques to comply with the strong constraints of tamper resistant hardware. Second, any advanced and potentially extensible database processing (e.g., large pieces of code implementing user defined database functions, stored procedures, database workflows or involving existing libraries supporting data intensive processes) is proscribed as it cannot be integrated as part of the TCB. These two main drawbacks drastically limit the practicality and the genericity of the PDS approach. The security is hence achieved at the price of extensibility. Hence, such solutions mainly target ad-hoc applications managing highly sensitive data, e.g., personal Electronic Health Records.

2.5. Synthesis of the existing approaches

The solutions presented above address different functionalities of the personal cloud and consider different trust models, which are both summarized in Tables 1 and 2.

---

[8] https://project.inria.fr/plugdb/en/

In terms of functionalities (see Table 1), two important conclusions can be drawn. First, **the whole personal cloud data life-cycle must be covered.** We observe that the different solutions tackle different stages of the life cycle of the personal data in a personal cloud. In particular, all solutions discussed above address *data collection*, *storage*, *backup*, *cross-computations and data dissemination*. An *extensive* personal cloud solution should hence include all these functionalities to cover the whole personal data life cycle.

Table 1. Main functionalities of the state-of-the-art personal cloud solutions

		Representative Personal Cloud approaches				
		Online personal cloud	*Zero-knowledge personal cloud*	*Home cloud software*	*Home cloud plug*	*Tamper resistant home cloud*
Functionality	*Storage*	Regular DBMS technology	Zero-knowledge cloud storage	Data stored on a generic user-side device, separated stores for different data sources	Data stored on a dedicated user-side device	Data stored in a tamper resistant device at the user-side
	Backup	Regular DBMS technology	Encrypted archive, point-in-time recovery	Replication / offline storage	Replication / offline storage	Replication / offline storage
	Data collection	Web scrapping	Files pushed or synchronized by the user to the cloud provider	Personal data inserted by the user or automatically to the home cloud	Personal files inserted by the user to the home cloud	Personal data pushed by the user or third parties to the home cloud
	Cross-computations	Transversal DB queries	At application level	Question answering, local data aggregation	At application level	Simple transversal DB queries
	Distributed computations					Simple distributed SQL statistics at large scale
	Data dissemination	[synchronization]	At application level	Minimum privileges for third parties and applications	[synchronization]	Minimum privileges for third parties and applications, secure access control

Second, **distributed computations should be part of the covered functionalities.** We also note that the *distributed computations* step is currently poorly covered. Is this because this functionality is less useful or because it is too difficult to be covered in practice in the personal cloud context? Regarding the utility of this functionality, we argue the opposite. Distributed computations over the personal data of (very) large sets of individuals unquestionably pave the way for Big –personal– Data computations with many applications in a personal cloud context, like computing recommendations, launching participative studies, learning information using the data of users belonging to a community (e.g., training a neural network in a patient community) or making collective decisions. However, this also requires privacy preserving implementations. A primary condition under which large sets of individuals would contribute with their own private data to collective uses is the guarantee that neither the other participants nor the infrastructure can access individual data. This probably explains why the only line of work addressing this step is focusing on security and proposes solutions based on tamper resistant hardware.

Trust is another essential concern of the personal cloud (see Table 2). Two conclusions can be drawn for the state-of-the-art analysis.

First, **all the privacy threats considered in the state-of-the-art solutions must be circumvented** to protect user's privacy and security in a meaningful way. Indeed, several threats are addressed by the different proposals, such as data snooping and secondary data uses performed by cloud providers (e.g., data monetization), corrupted applications or client devices (e.g., ransomware), or personal device failure. They all makes sense from a personal user point of view, since her whole digital life is managed and controlled using the platform.

However, a second (negative) conclusion is that **unifying these different solutions does not lead to a secure personal cloud architecture**. This is the case because building the union of the proposals would undeniably face irreconcilable architectural choices. Indeed, we observe that the existing personal cloud solutions cover a rather wide spectrum of architectural choices, but this leads to different – and sometimes contradictory – trust models and security measures. More precisely, each class of solutions addresses a specific subset of functionalities while

considering a specific threat model. For example, how is it possible to combine zero-knowledge encrypted storage with online personal cloud data-oriented computation facilities without returning all an individual's data to the client side and putting them at risk?

Table 2. Trust considerations in the state-of-the-art of personal cloud solutions

		Representative Personal Cloud approaches				
		Online personal cloud	*Zero-knowledge personal cloud*	*Home cloud software*	*Home cloud Plug*	*Tamper resistant personal server*
Trust	*Considered threats*	secondary usages (monetization) performed by the cloud provider	Data snooping or leakage, secondary usages performed by the cloud provider, client device failure and ransomware attacks	Massive data snooping or leakage, secondary usages, over-privileged third parties and applications	Massive data snooping or leakage, pecuniary usages, home plug device failure	Massive data snooping or leakage, secondary usages, over-privileged third parties and applications
	Trust model	Fully-honest personal cloud provider, trusted personal cloud code, trusted applications	Honest-but-curious to Malicious cloud provider, trusted applications, trusted client device (before time of failure or ransomware attack)	Trusted personal cloud code, trusted client device, untrusted applications	Trusted personal cloud code, trusted home plug, trusted client applications	Trusted personal cloud code, honest-but-curious central supporting infrastructure, untrusted applications
	Privacy and security measures	Security standards, virtuous legal and economic framework, transparency and auditability of the code (apps/PDMS)	Client-side encryption, 'no-knowledge' cloud store	SafeAnswers, logical separation of the personal data stores, audit	Closed platform (dedicated device), physical ownership	Secure hardware, physical ownership small TCB, secure distributed protocols

In conclusion, we can derive from this state-of-the-art analysis the set of functionalities to be implemented in the underlying Personal Data Management Systems (PDMS for short) to cover the complete data life-cycle, and the list of privacy threats the PDMS must circumvent. However, existing solutions do not address the whole functionality/threats spectrum and cannot be combined. Our goal in the next section is to progress towards a clearer definition of what an *extensive* (covering all the functionalities) and *secure* (addressing all the threats) PDMS should be.

3. Definition of an Extensive and Secure Personal Data Management Systems (ES-PDMS)

Currently, the principles underlying most existing solutions seem to be directly inherited from those considered in the context of the corporate cloud and have not been rethought with personal use in mind. For example, many solutions examined in the previous section rely on data encryption but nothing is said about restoring the master key in case of damage, except resorting on trivial unsecure protocols or on so-called-trusted third parties. Similarly, how to securely collect personal data from web sites through a myriad of unsecure wrappers without leaking both the user credentials needed to connect to the remote site and the collected personal data ? We argue that the way the PDMS functionalities are implemented and secured is determined by the intrinsic personal use of the PDMS, and must be deeply redesigned with this statement in mind.

In what follows, we first review in Section 3.1 each of the PDMS functionalities identified in the state of the art as needed to cover the whole data life-cycle and we analyze their intrinsic specificities. Then, in Section 3.2, we derive the fundamental security property attached to each functionality (and hence to each step of the data life-cycle). This analysis leads to the definition of an *Extensive* (i.e., providing the needed functionalities to cover the whole data life-cycle in a personal cloud) and *Secure* (i.e., achieving all the expected security goals) *Personal Data Management System* (ES-PDMS), which is provided in Section 3.3.

3.1. Specificities of data management in the PDMS context

As identified in Section 2.5, the PDMS functionalities are expected to cover the main stages of the personal data life-cycle and should thus integrate *data collection*, *storage and recovery*, *personal computations*, *distributed computations* and *data dissemination management*. For each functionality, we discuss their main specificities, and highlight to which extent they differ from their corporate data management system counterpart. Note that some

operational aspects (e.g., how to interact with the PDMS, how to engage into a collective computation, etc.) are more platform dependent and will be addressed in Section 4 where physical instances of PDMS will be discussed.

Data collection. The data collection functionality concerns both primary copies of user data (e.g., quantified-self data, smart home data, photos, videos, documents generated by the user, etc.) and secondary copies (e.g., banking data, health, employment, insurance, etc.). While the primary copies can be directly fed to the PDMS from data sources under the user's control, secondary copies have to be scrapped from the online services holding them. Collecting data from external sources is a basic operation of any corporate data management system. This task is usually handled thanks to a well-known and predefined set of carefully audited, patched and supported wrappers, under the control of data and security administrators who guarantee the quality and integrity of the integrated data. In the PDMS context, the situation is totally different. The PDMS owner is confronted with a large variety of scrappers (e.g., Web Outside of Browsers[9]) capable of capturing various types of data from a myriad of online services, the code of which cannot be trusted due to code complexity, diversity of contributors and sometimes closed source. However, by construction, such scrappers have access to highly sensitive data, from the credentials required to connect to the online service to the scrapped data itself (e.g., bank records, pay slips, invoices, medical records). Moreover, the user environment (e.g., operating system, network, other apps) in which the wrappers run is by far less trusted than an enterprise administered environment.

Storage and recovery. As any data management system, a PDMS has to securely store and ensure the durability of the data it manages. This means setting up encryption protocols to protect the data at rest against piracy and backup/recovery mechanisms to protect them against accidental loss. In a corporate DBMS, all these tasks are handled by DBA and DSA, having a recognized expertise in data management and security. Such expertise cannot be assumed for the PDMS owner. However, the risk of confidentiality and integrity attacks on private data has never been so high, as demonstrated by massive ransomware attacks affecting both individuals and organizations. Hence, the PDMS owner is facing the choice between endorsing the responsibility of data administration tasks that she cannot reasonably undertake or delegating these tasks to a (trusted) third party and thus abandoning the empowerment she just received from the PDMS paradigm. Zero-knowledge storage providers argue they have a solution to this problem, but this is by ignoring the issue of restoring the master key protecting the data in case of loss. They simply suggest deriving the master key from a password, but the password entropy is such that the cryptographic protection becomes highly questionable in this case. Cloud encryption is however not new and robust corporate solutions exist, like resorting on expensive Hardware Security Modules (HSM) to secure the master-key, a solution that individuals can unfortunately not afford.

Another distinguishing issue between the corporate and personal contexts is the liability regarding the hosted data. A company hosting personal data must guarantee the confidentiality of this data and can be subject to sanctions by a court in the event of a data leak. Thus, companies usually take appropriate measures to protect themselves. The legal responsibility of a PDMS owner remains unclear today, yet a PDMS can host personal data from many other people (e.g., contact details of doctors and relatives or external personal data gathered during a collective computation). Thus, the PDMS must protect this data on the owner's behalf and might even have to prevent her from accessing some of this data. Hence, the PDMS owner must not be granted access to the full content of her PDMS. Consequently, the PDMS owner must not even be granted a direct access to the master key required to decrypt the backup archive in case of a crash. This is a typical example of new issue raised in the PDMS context.

Personal computations. Personal computations in a PDMS usually refer to apps crossing various data of a single individual: the PDMS owner. We can distinguish between two types of such apps reflecting two specific uses of a PDMS: (i) apps used directly by the PDMS owner (e.g., for quantified-self, health and wellbeing, statistics and analyses related to smart home data or user's mobility, etc.); and (ii) apps representing external services to which the owner willingly subscribes (e.g., billing apps allowing a car insurance company to compute the premium based on the owner's car trajectory data – as in pay-as-you-drive, or an electric company to compute the bill based on the user's electric smart meter traces). Therefore, an important specificity in the PDMS context is that apps "move" towards the data as opposed to personal data migrating towards remote services as it happens with most existing cloud services.

This has two main implications. First, the apps manipulate sensitive raw data, but neither the apps nor the environment in which they run can be trusted in general, leading to similar security problems as the ones discussed

[9] weboob.org

for data collection. *By-default auditing mechanisms* are thus required to detect malicious apps deviating from their manifest. These mechanisms must be easily understandable by non-expert users, disqualifying advanced audit tools based on complex models and formal languages usually employed by audit experts and security administrators in an enterprise context. Second, some external service apps need strong guarantees regarding the results produced by a PDMS (e.g., the billing apps discussed above). That is, an *attestation* process is required to ensure that the result was indeed produced by a certain computation code using all the required input data, and that the PDMS owner cannot tamper with the computation process nor the inputs.

Collective computations. Collective computations relate to various types of big personal data processes computed over a large set of PDMSs (e.g., contributing to participative studies, training a neural network, building an anonymous dataset). In addition to the security issues already discussed regarding the intrinsic untrusted nature of apps and computing environment, collective computations introduce a new difficulty. Gathering all the participants' data in a single place to perform the computation introduces a single point of vulnerability and maximizes the incentive to attacks. Conversely, decentralizing the processing implies to temporarily transfer personal data among participants, transforming each into a potential attacker. In this latter case, two guarantees must be provided: (i) data confidentiality, i.e., any PDMS owner cannot access the data in transit of other participants, and (ii) distributed computation integrity, i.e., any participant PDMS can attest that any result it supplies corresponds to the assigned computation. Classical distributed computation techniques used in enterprise systems cannot apply here due to the unusual scale of the distribution (i.e., the computation may target a fraction of the population of a country). Generic secure multiparty computation protocols based on cryptographic techniques (MPC) are disqualified for the same reason (performance does not scale with the number of participants). Conversely, the participants cannot trust each other since participants are unknown a priori (and probably wish remaining anonymous).

Data dissemination management. The purpose of a PDMS is to enable the owner to make the necessary decisions regarding the dissemination of his or her personal information. In particular, according to the aforementioned functionalities, the user should be able to give the appropriate permissions on the data or documents to share with acquaintances and distant third parties, to decide which personal computations she will authorize and which collective computations she accepts to contribute to. In a corporate context, such decisions would be managed and enforced by central authorities and would once again rely on IT experts (DBA and DSA) who define appropriate roles, set access control policies (e.g., following RBAC, MAC, ABAC or TBAC models), and provide system security and audit to ensure that everything goes as planned. On the contrary, the PDMS context puts such decisions and their enforcement into non-expert users' hands, with the risk of generating more security holes than solving them. For example, the aforementioned access control models are not well adapted to a non-expert administrator having to manage a highly dynamic set of interactions with a myriad of other users and third parties. Specific tools have been suggested to let individuals manually define their sharing preferences (e.g., thanks to PGP, Web of Trust models or FOAF dissemination rules) but they provide little consistency guarantees about the final outcome. Conversely, relying on a trusted third party to manage personal data dissemination would be contrary to the very notions of user control and empowerment. Hence, whatever the way the PDMS owner defines her sharing preferences, the PDMS must provide ad-hoc tools to help her easily understand the net effects of her decisions related to data dissemination and sanitize the policy accordingly when required. In addition, the PDMS owner is not a super-user having all privileges over the full content of her own PDMS content. As already stated, a PDMS may host other users' personal data and the PDMS must protect this data against unintended actions of the owner of the PDMS herself.

3.2. Security properties of a PDMS

As a conclusion of the preceding analysis, the PDMS context sketches an open and rich ecosystem of new untrusted data processing apps in interaction with an unsecure execution environment and a layman PDMS owner. This significantly contrasts with corporate data management systems where the applications and computing environment are significantly more static and carefully controlled by data and security administrators. Additionally, typical distributed computation infrastructures (cluster/cloud) strongly differ from a fully decentralized infrastructure of PDMSs (e.g., in terms of scale, ownership, legal agreements, deployed hardware and software, etc.). As a consequence, the PDMS must integrate by default novel security measures to overcome the inherent weaknesses of the PDMS owner and tackle the specific threats to this open and untrusted ecosystem. We detail below the security properties expected from a PDMS, and linked to each functionality described in the previous section (one security property is thus associated with each step of the data life-cycle). We make no assumption about the technical means to enforce these properties, delaying this discussion to the next sections.

Piped data collection. Under the hypothesis of untrusted collection code and untrusted user computing environment, a PDMS is said to enforce *piped data collection* iff:

1. the only PDMS data accessible by the collection code are the credentials allowing access to the related data providers;
2. the credentials and the collected data related to a given data provider cannot be leaked outside the PDMS and the data provider.

   This property guarantees that the only channel to the outside world provided to the data collector is a specified data provider and that the code is suitably isolated so as not to be able to leak data to a potentially corrupted user environment.

Mutual data at rest protection. Under the hypothesis of a layman PDMS owner, a PDMS is said to enforce *Mutual data at rest protection* iff:

1. the PDMS unconditionally protects the hosted raw data and the backup archive against any form of confidentiality and integrity attacks or accidental damages conducted by external adversaries or by the PDMS owner herself;
2. The secret protecting the backup archive is recoverable;
3. This secret is not accessible to the owner nor any other party except another PDMS belonging to the owner and providing all the expected PDMS functionalities and security properties (typically, an *Extensive and Secure PDMS* as defined in Section 3.3).

Since a PDMS stores raw data from the owner and also personal data from other users, data protection must also operate against the PDMS owner, thus the term *mutual*. To be effective, the PDMS must enforce this property automatically, without any owner intervention which could open the door to administrator attacks. Moreover, it requires that the archived data can only be interpreted by a PDMS, whatever the way the secret protecting it is produced, made resilient and recovered.

Bilaterally trusted personal computation. Under the hypothesis of untrusted external code and untrusted owner computing environment, a PDMS is said to enforce *bilaterally trusted personal computation* iff:

1. a personal computation may access only the owner's raw data specifically required for the computation;
2. only the final result of the computation - not the raw data - may ever be exposed to a third party;
3. the execution of the computation produces trustworthy audit trails accessible to the owner;
4. the PDMS can provide a proof that the result of the computation was produced by the expected code.

This property provides *bilateral guarantees* to the PDMS owner and the third party willing to execute code on the owner's data. It guarantees to the former that the minimal collection principle enacted in laws protecting personal data (e.g., GDPR) is fulfilled, that the computation cannot leak unexpected data and finally that she will have the ability - not the obligation - to audit the compliance to this property. Conversely, it guarantees to the latter (e.g., an energy provider willing to compute the owner's bill) that the code remotely sent to the PDMS has been accurately computed. If this computation combines several tasks, the proof produced by the PDMS must guarantee that the orchestration of these tasks cannot be tampered with without the caller being able to detect it. However, the PDMS cannot attest by itself that the data targeted by this code is genuine. Such attestation remains under the responsibility of the computation code, assuming that the data has been properly signed by their producer.

Mutually trusted collective computation. Under the hypothesis of untrusted external code and untrusted user computing environment, a PDMS is said to enforce *mutually trusted collective computation* iff:

1. a collective computation may access only the participants' raw data specifically required for the computation;
2. only the result of the computation - not the raw data - may ever be exposed to a third party or to any participant;
3. the execution of the computation on a participant generates trustworthy audit trails accessible to that participant;
4. a proof can be provided that the result of the computation was produced by the expected code over the expected set of participants.

This property targets the same objective as its bilaterally trusted personal computation counterpart. It must

integrate the fact that participants can contribute to the collection phase and/or the processing phase of the computation with no assumption on the cardinality of these two sets of participants and on their intersection.

***Controlled data dissemination*.** Under the hypothesis of a lay PDMS owner and of an untrusted execution environment, a PDMS is said to enforce *controlled data dissemination* iff:

1. the integrity and confidentiality of interactions between the PDMS and its owner are guaranteed, when defining the dissemination policy and auditing its effects, and when decisions are made regarding the regulation of data dissemination
2. the decisions are enforced by the PDMS and cannot be circumvented;

This property guarantees to the PDMS owner that all decisions (e.g., entrust a secret share to a remote user for recovery purposes or allow a collective computation) are faithfully captured (point 1), which calls for integrity guarantees to ensure that the decisions cannot be corrupted when captured, and confidentiality to ensure that the decisions themselves and the personal data on which they may rely cannot be leaked. Thus, the effects of these decisions in terms of data dissemination are enforced by the PDMS and cannot be circumvented (point 2) neither by any third party, nor by any potentially untrusted parts of the execution environment, and not even by the PDMS owner who may have restricted privileges (e.g., over data generated by other users, or over cryptographic secrets or metadata generated for administrative purposes such as ensuring mutual trust as explained above). Finally, audit tools (point 1) are provided in order to help layman PDMS owners understanding all the effects of the taken decisions in terms of data dissemination and allow her to act to update decisions or rectify their effects if needed.

3.3. Extensive and Secure Personal Data Management Systems

The definition of an *Extensive and Secure Personal Data Management Systems* is directly derived from the previous sections and is expressed as follows:

***Extensive and Secure PDMS*.** An *Extensive and Secure Personal Data Management Systems* provides the expected set of functionalities to cover the complete data life cycle in a personal cloud, namely *data collection*, *storage and recovery*, *personal cross-computations*, *collective computations* and *data dissemination management*, and is compliant with their respective security properties counterparts, namely *piped data collection*, *mutual data at rest protection*, *bilaterally trusted personal computation*, *mutually trusted collective computation* and *controlled data dissemination*.

4. Extensive and Secure PDMS Architecture

Designing an extensive and secure PDMS architecture providing the functionalities and the five security properties defined above is highly challenging, given the fundamental tension between the security expectations of a layman PDMS owner and the need for supporting an open ecosystem of applications running on an untrusted environment. In this section, we introduce the fundamentals of a PDMS reference architecture tackling this tension and detail its building blocks. We then show that physical instances of this reference architecture can be already envisioned today and discuss how existing and forthcoming software and hardware mechanisms may impact the satisfaction of our security properties.

4.1. Logical architecture

Ideally, the support of potentially complex manipulations of personal data while preventing unexpected data leaks could be achieved by securely collecting, storing and manipulating the data in a secure subsystem, managed under the control of the holder, while never letting the (untrusted) applications or third parties directly access the raw data. Such a clean separation can only be based on the assumption that there exist a few generic functions that can be used to manipulate personal data without violating privacy. Such an assumption is obviously a fantasy, because the most interesting manipulations of personal data are application-specific and consequently, privacy violations are also application-specific.

To solve this problem, we propose a three-layer logical architecture where a minimal *Secure Core* (Core) implementing basic operations on personal data is extended with *Isolated Data Tasks* (Data tasks) themselves accessed by *Applications* (Apps) on which no security assumption is made (see Figure 1). The objective is to control the flow of raw personal data from the Core to the outside, such that only expected results are declassified to untrusted applications or third parties. The general description of the architectural layers follows:

- *Core*. The Core is a secure subsystem that is a Trusted Computing Base (TCB) ideally minimal, inextensible, proven correct through formal methods and isolated from the rest of the system. The Core must provide all basic operations required to enforce the confidentiality, integrity and resiliency of the personal data hosted by the PDMS. It must be the unique entry point to manipulate this data. The Core must thus implement a *data storage* module. A *policy enforcement* module must be integrated in the Core to regulate the data access performed by the other layers of the architecture. A *communication manager* is also needed to securely communicate with other users, applications and third parties. In what follows, we give an empirical view of the Core minimality, by identifying which combination of functionalities is (strictly) mandatory in each of the three parts of the Core to guarantee the security properties introduced in Section 3.2.

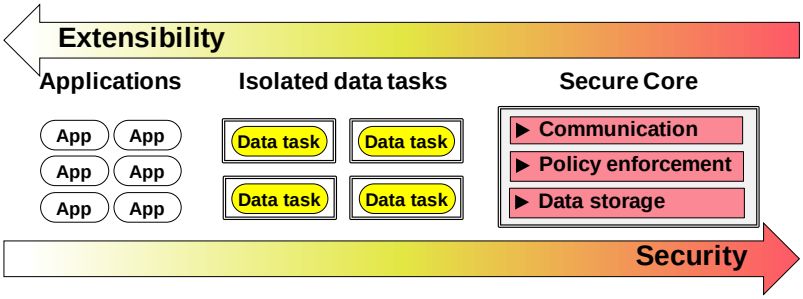

Figure 1. Global Architecture

- *Data Tasks*. Data tasks are introduced as a means to deal with application-specific personal data management. 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.
- *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 (web browser, smartphone, etc.) 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.

4.2. Building blocks

This section details how each security property introduced in Section 3.2, namely *piped data collection*, *mutual data at rest protection*, *bilaterally trusted personal computation*, *mutually trusted collective computation* and *controlled data dissemination*, can be practically satisfied. For each property, we identify the required elementary building blocks and explain how they should be combined to reach the expected goal without introducing security breaches.

Each building block in turn relies on a set of *common security primitives* provided by the Operating System (OS) and/or the hardware platform hosting the PDMS. Hence, these primitives are the foundations of our empirical minimal definition of the Core. Since they are commonly used by various building blocks, we present them first. As their implementations differ across platforms, we concentrate below of the security primitives they provide.

***Common security primitives*:**

- *Isolation*. A component of the architecture is said to be isolated if (i) the internal execution state of the component cannot be accessed nor influenced from the outside of the component except with the collaboration of the system administrator (e.g., the PDMS owner) and (ii) the component may not observe nor influence the behavior of any external system except through its own inputs/outputs behavior. See for example [28] for a survey of ways to implement code isolation in a partly untrusted context.
- *Attestation*. A component is said to be attestable if a trustworthy certificate can be produced to demonstrate that the component output was indeed produced by the specific code of this component. This common security primitive is usually considered for a whole complex system (e.g., [21]). It was formalized in the case of a single task running on a complex system in [16].

- *Confidentiality*. A component is said to ensure data confidentiality if neither the internal execution state nor the input and output data of the component can be leaked to any system other than the party initiating the component (in our case the PDMS Core) even with the collaboration of the system administrator. This property is described in [32] and formalized together with isolation and attestation as 'secure outsourced computations' in [16].
- *Peripherals isolation*. A component is said to satisfy peripherals isolation if the data exchanged between that component and the peripherals cannot be leaked outside of the component except with the intervention of the PDMS owner. For illustration purpose, [7] and [36] show how text messages can be securely displayed even in the presence of an untrusted OS using ARM TrustZone.

For the sake of clarity, each the security primitive defined here will be represented by a simple pictogram in the next figures showing the building blocks required to implement each of the security property introduced in Section 3.2. The pictograms used are ▣ for code isolation, 🏅 for attestation, 🔒 for confidentiality and 🖳 for peripherals isolation.

4.2.1. Piped data collection

The piped data collection property requires the ability to execute arbitrary data collection code (e.g. a scrapper) in a secure manner. The objective of this property is actually twofold.

First, it should guarantee that the collection code will not access any data stored in the PDMS other than the credentials required to connect to the related service provider (e.g., the web site to be scrapped). This specific privilege must be part of the manifest declared at the time this collection code is registered in the PDMS. The resulting authorization itself will be enforced at execution time thanks to the controlled data dissemination property (see Section 4.2.5 for details).

Second, it should guarantee that the collected data and the credentials related to a given data provider cannot leak to any third party (including another data provider targeted by the same collection code). According to the reference architecture sketched in Figure 1, this guarantee can be provided by considering the collection code as an isolated data task and granting a write access to this data task only to the destination PDMS. This requires data task *authentication* to be implemented in the Core. In other words, this means restricting the write capacity of the data task to the insertion of the collected data into the Core. The enforcement of this restriction at execution time relies itself on the code isolation property.

While executing the collection code as an isolated data task inside the PDMS ensures that the effects on the Core are controlled, further measures are required in order to ensure the absence of leakage of both credentials and collected data. Indeed, the collection code needs to communicate with the outside world in order to reach the data provider. To preclude the collection code to leak data to any other party than the related data provider, the Core must be able to establish a (TLS for example) secure channel between that specific data provider and the data task. Regarding the minimality objective, the complete network stack (e.g., TCP/IP, DNS) does not have to be in the Core, only the critical security operations of the creation of the secure channel, named *TLS trusted* in Figure 2, need to be.

Remark that each data collector task should be dedicated to a single remote site (e.g., my bank), to avoid a malicious data task from leaking credentials or personal data (e.g., the bank credentials/data to another site) through an authorized communication channel. Note that in addition a malicious data collector task may use the owner's credentials on the remote site to perform unexpected actions (e.g., the data collector retrieving the bank related data could trigger a money transfer using the bank credentials). However, such issues are mostly related to the definition of a weak security policy at the data provider side, rather than a problem to be addressed at the PDMS architectural level. We thus assume here that the credentials delivered by the data provider for data collection purposes will grant only read access to the reduced dataset of interest (e.g., bank account history).

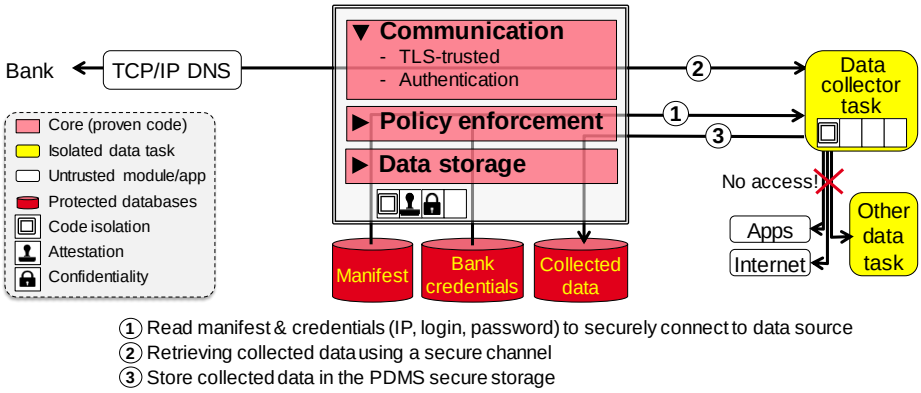

Figure 2. Data collection

Summing up the architecture presented in Figure 2, when performing data collection for a specific source, the Core launches an isolated data task executing the collection code for said source, provides it with the credentials and a secure channel to the source, which requires the implementation of data task authentication and secure channel set up (TLS-trusted) in the Core. Once collection is finished the collector returns the data to the Core which stores it appropriately. This ensures the absence of leakage (through isolation and secure channel), and proper behavior of the data collector in terms of input and output data (through access control). Another statement is that the safety properties are not independent from each other. Piped data collection indeed relies on controlled data dissemination and mutual data at rest protection to make sense when considered in a complete scenario.

4.2.2. Mutual data at rest protection

This property should be ensured by the Core which is the only entity authorized to have a full access to the PDMS data (recall that, even the owner does not have such an access, since the PDMS data may include, for instance, data from other users). *Secure storage* is thus ensured by the Core, potentially using classical cryptographic techniques (encryption, hashing) if the storage medium is not part of the Core, to protect the data against confidentiality and integrity attacks. In the following, we focus on the backup and recovery issues which are less classical in the PDMS context. We note that the secret protecting the backup archive can obviously not be stored on the PDMS (otherwise, it would be lost in case of failure). As already mentioned, relying on a password is not adequate due to the generally reduced entropy and to the difficulty and the risks associated with the memorization of complex passwords, with no way to reinitialize it. In addition, the owner should not be capable of restoring this secret alone since he does not have full access to the PDMS data, further disqualifying password-based solutions. A reasonable solution to this problem is to split the secret in a number of secret shares using a secret sharing scheme (for instance [2]). We provide below a simple example of how such a task could be performed, using standard cryptographic techniques and the *common security primitives* introduced at the beginning of Section 4.2.

We split the backup and recovery process in three steps. First the setup, which consists in generating and distributing all cryptographic material necessary for performing the encrypted backup and recovery. Then comes the backup phase. Finally, assuming one's PDMS has been lost/compromised, comes the recovery phase.

Setup phase:

1. The PDMS generates a master key, inaccessible to the PDMS's owner, which will be used for performing encrypted backup
2. The PDMS owner chooses a number *n* of trustees (among her friends/family), who will hold shares of her master secret key. She also chooses a security threshold *s*. This threshold is the number of trustees who need to cooperate to recover the owner's PDMS content.
3. The PDMS executes an *s* out of *n* secret sharing scheme (e.g., Shamir's secret sharing). The shares are subsequently distributed through a secure channel (using the TLS-trusted module) to the PDMSs of the trustees. We need to ensure that secure channels are indeed established with the trustees' PDMS and no other parties (i.e. to avoid other people from getting the shares). This can be done using a certificate that proves that a PDMS is genuine or using the Core hardware attestation mechanism of the PDMS if available. Note that it is essential that the trustees

themselves cannot access the share directly, even though it is stored in their PDMSs. Indeed, if they could access these shares, they could, in collaboration with the PDMS owner, decrypt all encrypted backups outside a PDMS, and thus access data that is meant to not be accessible to the PDMS owner herself. Therefore, the permissions for these shares need to be set to only be accessible to the recovery functionality.

Backup phase: this phase is not specific to the PDMS context. The encrypted backup of the PDMS is performed on an untrusted third-party server by the backup module of the PDMS. One should be careful to choose a backup scheme that preserves integrity as well as secrecy (e.g., SpiderOak backup service [26], see Section 2.2).

Recovery phase: This phase is triggered when the user has lost access to its PDMS and needs to obtain a new copy of her data. It proceeds as follows (note that the practical means to realize this protocol may be adapted to minimize the PDMS's owner burden).

1. The user acquires a new empty PDMS.
2. The user contacts *s* trustees, and informs them that they need to perform the recovery procedure, with the identification data of the new PDMS, using an out-of-band communication channel (e.g., phone call, chat or email message).
3. The PDMS recovery modules of the *s* trustees communicate their shares of the master secret to the recovery module of the new PDMS, using the provided identification data of the new PDMS, on a secure channel (using the TLS trusted module again). It is essential to ensure that these shares are indeed communicated to a genuine PDMS, belonging to the user. As in step 3 of the setup phase, to achieve this guarantee, either a certificate or remote attestation is used by the trustee's PDMS in order to ensure that they are indeed communicating with the recovery module of a specific PDMS.
4. The recovery module of the new PDMS reconstructs the master secret using the share provided by the trustees' PDMSs.
5. The new PDMS retrieves the encrypted backup and recovers the data using the master secret.

As illustrated in Figure 3, the main impact on the architecture is the need for *backup and recovery* modules in the Core. As the recovery can only be performed inside a legitimate PDMS, the user is never given direct access to restricted data. Additionally, neither the user nor the trustees may gain access to the secret shares, which ensures that the encrypted backup may never be decrypted outside the recovery process. Finally, the *s* out of *n* secret sharing ensures that the owner's secret is recoverable, even if the PDMS of a number of trustees fail. One can choose the threshold according to her confidence in the hardware and the reliability of her trustees and may even include one or several mandatory trustees in the secret recovery process.

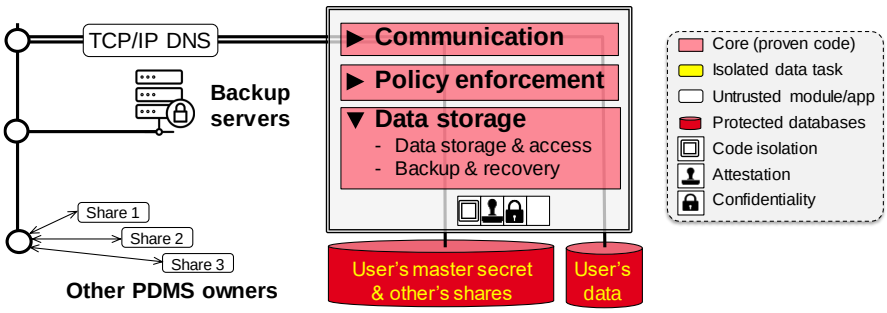

Figure 3. Mutual data at rest protection

4.2.3. Bilaterally Trusted Personal computations

First, as required in the security property (see Section 3.2) the personal computation functionality needs to be able to execute (arbitrary) code ensuring that only raw data required for the computation is made available to the code. The minimality requirement on the Core prohibits the execution of such complex tasks inside the Core. As a consequence, extensibility in terms of code execution is achieved by executing personal computations as data tasks. In order to restrict access to raw data, we require that any personal computation task comes with a manifest specifying precisely the data needed for said computation. This manifest should be approved by the user (see the controlled data dissemination property in Section 4.2.5). Subsequently, when executing a personal computation

task, the Core's *reference monitor* provides only the data required by the manifest, ensuring by construction that the manifest is respected.

Second, only the result of the computation should be made available to third parties. As the user's system may be corrupted, this implies that the personal computation task should be isolated from the environment. This prevents the user's system from accessing the internal state of the data task, hence preventing leaks of raw data. Additionally, the data task should be authenticated by the Core and only be able to store the result in the Core. Any subsequent disclosure of the result to the outside world should be done by the Core and subject to control from the *reference monitor*.

Third, the computation should provide an audit trail accessible by the owner. This is again achieved through the *reference monitor*, which should update the audit trail accordingly through the *audit* component of the Core. In order to guarantee the minimality of the *audit* component of the Core, only simple actions should be logged, such as a handle on the data used in the computation and a handle on the result together with a handle pointing to the manifest of the data task which has computed the result. This avoids providing the complex relationship between the data and the result directly in the log while retaining the ability to reconstruct this relationship from data safely held in the PDMS.

Finally, it should be possible to provide a proof that the result is indeed produced by the expected code. This is achieved through *attestation* of the data task which exactly provides this guarantee. Note that, while *attestation* provides guarantees that the result was indeed produced by a specific computation task, this does not in any way provide guarantees on the raw data that was used in order to compute said result. If one wants to obtain guarantees on the data used, the checks should be included in the personal computation code. For example, if an energy provider wants to compute the monthly energy consumption from certified data from the energy meter using a personal computation task, this computation task should include checking the certificates for the data; otherwise the user might execute the data task on counterfeit data and the attestation would still certify that the result transmitted was indeed produced by the right computation task.

In order to perform computations that are not represented by one atomic task but rather by a succession of tasks, the task can leverage attestation in order to provide guarantees on the end result in a manner similar to checking certificates for data. Indeed, if a computation task *T* is supposed to be executed on some data resulting from the execution of a previous computation *R*, *T*'s code can verify that its input data is attested as the result of task *R*. Using this mechanism iteratively, it is possible to guarantee the integrity of a result coming from an arbitrary combination of computation tasks.

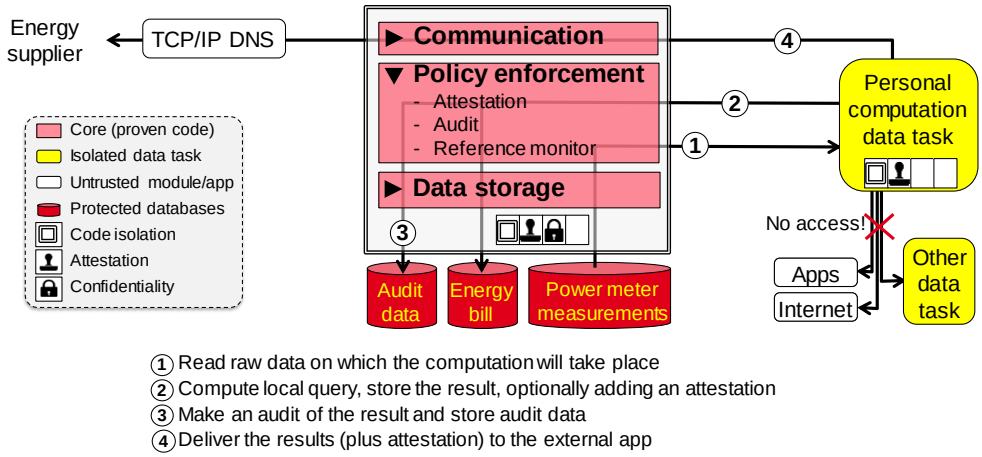

Figure 4. Bilaterally trusted personal computations

This functionality is presented in Figure 4. To sum up, this implementation of the functionality ensures that only the required data is accessed (through *reference monitor*), that this data may never be leaked to the outside world (through isolation), that the operation flow is auditable by the user (through the *audit* component), and finally that a proof can be provided that a result corresponds to a specific computation (through *attestation*). This leads to the introduction of a *reference monitor*, an *attestation* module and an *audit* module in the Core.

4.2.4. Mutually trusted collective computations

This property targets the same objective as its bilaterally trusted personal computation counterpart. It must integrate the fact that the participants can contribute to the collection phase and/or the processing phase of the computation with no assumption on the cardinality of these two sets of participants and on their intersection.

As for the personal computations, our architecture implements collective computations as data tasks (or more precisely as a set of data tasks communicating through the network) in order to achieve system extensibility. The *reference monitor* ensures that a collective computation data task can only access the data required for the computation (point 1 of the *Mutually trusted collective computation* property in Section 3.2). Note that this data may include intermediate results transferred by other PDMSs participating in the collective computation. As in the personal computation case the data access requirements are specified by the manifest of the collective computation, which has to be accepted by the user. Also, to enable the PDMS owner to audit her computations (see Section 4.2.5), the access audit module records information related to the execution of collective data tasks (point 3 of the *Mutually trusted collective computation* property in Section 3.2).

Another requirement of collective computation (point 2) is that only the result of the computation can be disclosed, and not the raw data. To this end, a first measure is to execute the collective computation data tasks as isolated data tasks. This protects a participant's raw data against an untrusted system environment. However, a collective computation requires the transmission of intermediate results between data tasks belonging to various participant's PDMSs. In order to ensure the confidentiality of these intermediate results, the *reference monitor* takes the following measures. First, it only gives access to the intermediate results to the other PDMSs executing the collective computation which are allowed to get these results as specified in the respective computation manifest. Second, at local level, it only gives access to the intermediate results to the specified data task and forbids any access by the PDMS' owner. Hence, to guarantee the enforcement of access control against malicious PDMS owners, the collective data tasks have to implement the confidentiality security property. Also, transferring intermediate results to other participating PDMSs is done using the *TLS-trusted* module to ensure that data is transmitted on a secure channel.

The final requirement for collective computation (point 4) is to ensure that a proof can be provided that the result of the computation was produced by the expected code running on the expected set of participants. This requires a global manifest (i.e., a formal description of the computation including the participants and their delegated tasks) allowing to check that messages originating from other participants were produced as expected. The global manifest is registered in the Core of each participant involved in the collective computation. We then need to propagate trust between nodes during the execution of the global computation protocol. Therefore, each data task output involved in the distributed protocol should be certified through attestation, such that each Core can check the local integrity of its local data task, and propagate it to the other participating Cores in order to incrementally establish the global integrity of the distributed protocol, with acceptable performance. Note indeed that instead of being verified by the third party exploiting the result as it is the case for local computations (see Section 4.2.3), the attestation must be verified here by the participants involved in the collective computation.

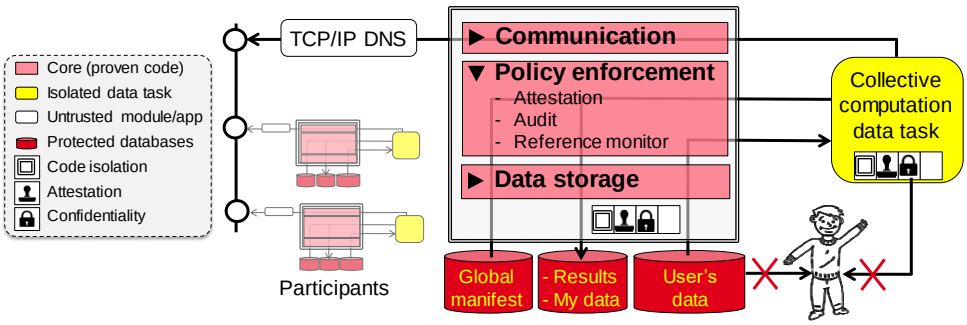

Figure 5. Mutually trusted collective computations

In conclusion, mutually trusted collective computations (as defined in Section 3.2) can be implemented in the proposed PDMS logical architecture by using isolated and confidentiality protected data tasks which are controlled and attested by the Core. As shown in Figure 5, the main modules needed for this type of computations are the *reference monitor* and *audit* modules to ensure that only proper data access is performed, and the *attestation* module to obtain global integrity guarantees for the computation.

4.2.5. Controlled data dissemination

A PDMS must provide a secure way to capture the decisions of the owner, accordingly implement and enforce the resulting effects on data dissemination, and help users to understand these effects and apprehend what effectively happens during data dissemination. Of course, the overall process is highly complex and opens up several challenges (see Section 5, Challenge 2).

Our goal here is to introduce the main architectural elements induced by *controlled data dissemination* and guarantee the two points of the corresponding security property as defined in Section 3.2.

First, securing a decision impacting data dissemination forces the PDMS to provide means for users to truthfully view the needed information to make the decision (e.g., view a manifest describing a computation, the underlying personal data to be authorized, etc.), capture it (at the very least, a button to accept/decline that manifest) and potentially check its effects and/or audit its effective use. From a security viewpoint, this means relying on the use of visualization software (e.g., file or image readers, up to more complex visualization tools according to the semantics of the manifest or the audit trails), which may contain external code with security breaches that could be exploited by an attacker. Therefore, this code cannot be part of the Core, but should be executed as part of a data task, called here a *decision-making* data task. In addition, in order to be sure that the code of this data task behaves as expected, the integrity of the execution should be guaranteed against any potentially malicious external entities or corrupted runtime environments. Moreover, the decisions which are made may also be considered personal for the PDMS owner or depend on personal information, and should thus not be leaked outside the PDMS (except through a deliberate intervention of the owner). To ensure both the integrity of the execution and the confidentiality of the decision, decision making data tasks must be run in *isolation*. Beyond this, making decisions by nature relies on interactions with the PDMS owner via peripherals (e.g., screen, keyboard). This requires *peripherals isolation* to ensure that the data exchanges between the peripherals and the data task cannot be altered nor leaked outside the data task. Subsequently, in order to correctly interact with decision making data tasks, the Core must ensure that the data task effectively runs the expected (original) version of the code. This requires authenticating the decision-making data task through the *authentication* module integrated into the Core (already introduced above).

Second, all the decisions must be unconditionally enforced. The decisions captured by the PDMS are thus translated into low level data dissemination policies and enforced by the Core. Enforcing decisions typically leads in our system to implement access control policies managed by the access control module and relying on a trusted *reference monitor* to enforce the effects of these access control policies. Of course, the form of the policies (e.g., access control lists, execution privileges on computations, etc.) and their use typically depend on the decision to be enforced.

We argue that the overall architectural design presented on Figure 6 offers an interesting backbone to operate data dissemination decisions in the PDMS context. This requires a *reference monitor* to store and enforce decisions, and an *audit* module to store audit data on the effects of these decisions. We give two example below showing how controlled data dissemination could be operated in the case of the recovery protocol described in Section 4.2.2 and in the case of local computations presented in Section 4.2.3:

Controlling data dissemination in data recovery. When a PDMS owner *Alice* wants to share a secret share *S* with another PDMS owner *Bob*, a decision-making data task with access to the contact files of *Alice*'s PDMS is launched, is authenticated by the Core of *Alice*'s PDMS and displays a list of *Alice*'s contacts from which she chooses *Bob* as a recipient of the secret share *S*. The effect of that decision on the *reference monitor* of *Alice* is to grant a read privilege on *S* to *Bob*'s PDMS. On *Bob*'s PDMS, *Bob* makes the decision to accept *Alice*'s secret share *S* (using a *decision-making* data task). As a result, *Bob*'s *reference monitor* grants a read permission on *S* to the *recovery* data tasks (no other data task nor *Bob* himself have granted access to *S*). At recovery time, *Alice* contacts *Bob* (out-of-band communication) and *Bob*'s PDMS contacts *Alice*'s the new PDMS. *Bob* accepts his PDMS to declassify *S* to Alice's using a decision-making data task and assuming that *B* confirms to its PDMS that the recipient of *S* is indeed the new PDMS owned by *Alice*.

Controlling data dissemination in local computations. Alice wants to authorize her energy supplier *EnergySupp* to run a data task *consum* which aggregates the electricity consumption of *Alice* on the last 30 days. A *decision-making* data task shows the manifest of *consum* to *Alice* which includes a description of the raw energy data taken as input (energy trail of the last 30 days, called *raw-data-30*). *Alice* accepts it and the resulting effect on *Alice*'s *reference monitor* are an *execute* privilege on *consum* granted to *EnergySupp*, a read privilege on *raw-data-30*, a

write privilege on its result to *consum*, and a read privilege on the result to *EnergySupp*. *Alice*'s PDMS also audits the execution of *consum* when it is launched by *EnergySupp* and the access to its successive results by *EnergySupp*. *Alice*, when looking at the audit trail realizes that *EnergySupp* accesses *consum* every day and is thus able to infer her energy consumption on a daily basis. In consequence, *Alice* may either revoke this decision or apply a 'fix' (which could have been provided by *EnergySupp* or by a community of users) to update the read privilege on *raw-data-30* of *consum* such that only a fixed size batch of data is included in *raw-data-30* (e.g., the data of the previous month).

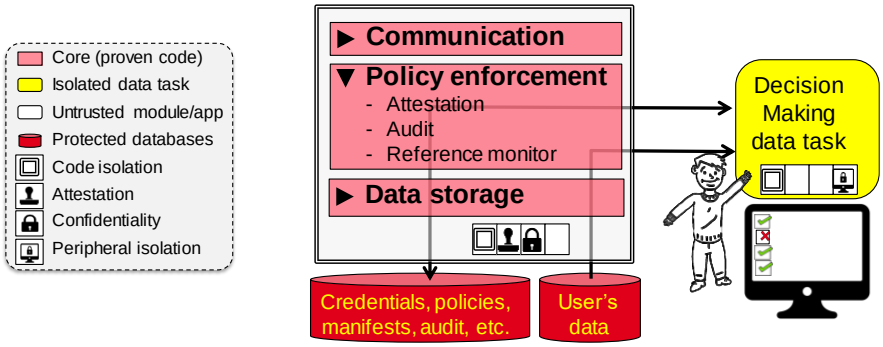

Figure 6. Controlled data dissemination

4.3. Overall architecture design

The union of the above-mentioned building blocks constitutes a baseline for a logical architecture to manage the data life cycle in a personal cloud, while answering the main security goals.

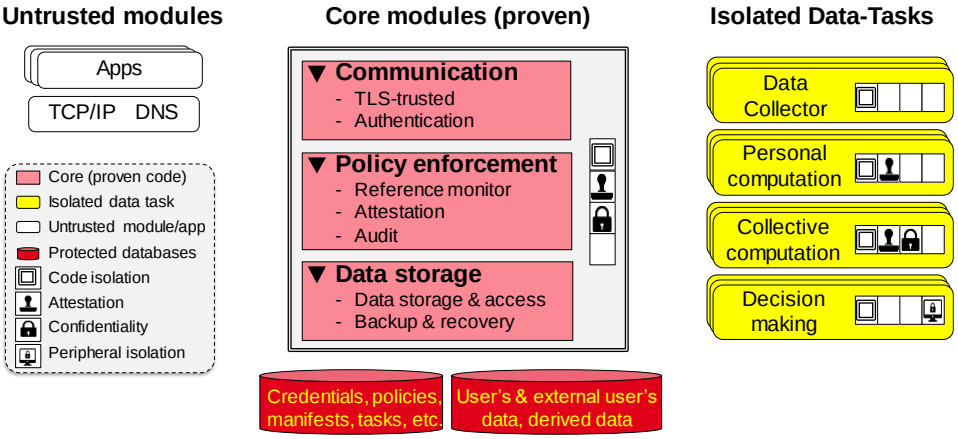

Figure 7. Logical architecture

Depending on its type, each data task requires an execution environment that provides the appropriate *common security primitives* introduced in Section 4.2, namely *Isolation*, *Attestation*, *Confidentiality* and *Peripherals isolation*.

Ideally, the code of the Core must be formally proven to avoid security breaches leading to unexpected behaviors. Equally important, the Core must run in an execution environment that satisfies isolation, attestation and confidentiality. The isolation property is required for the Core to protect it from all the other software components running on the same personal cloud platform (in particular the Apps and the data tasks). The attestation property is required for the collective computations which are orchestrated and/or executed by the Core. Finally, to provide mutual guarantees of security between PDMS users and third parties (see Section 3.2), the environment of the Core also has to provide confidentiality since it coordinates the distributed data tasks with potential access to private personal data supplied by other nodes, which remain hidden from the PDMS owner. This leads to the architecture presented in Figure 7.

4.4. Concrete PDMS instances

The goal of this subsection is to show that the logical architecture presented above can be instantiated in practice, using existing software and hardware solutions, and that its modular aspect helps defining physical PDMS instances easily. We first discuss here existing software and hardware security solutions, offering the required security primitives (namely *isolation*, *attestation*, *confidentiality* and *peripheral isolation*). Second, we show how to combine them into physical ES-PDMS architectures which instantiate three different configurations: (1) PDMS on a home box, (2) on a mobile device and (3) in the cloud. Third, we provide an example of a preliminary implementation of the proposed architecture.

There is no unique way of implementing the common security primitives on which our architecture relies, but rather various solutions with different guarantees regarding the enforcement of these primitives. We investigate below existing software and hardware targets.

Software-based security solutions. Let us first consider the security properties which could be provided by pure software-based solutions. The first of these properties is *isolation*, which is intensively investigated as it constitutes a foundation of secure software architectures. Isolation is usually provided by an operating system (e.g., Linux, seL4, etc.), a virtual machine monitor (VMM or hypervisor, e.g., XEN, KVM) or a container manager (e.g., Docker, Kubernetes). Such solutions have the advantage to provide a high level of *extensibility* (in the sense that potentially complex/external code can be run). But enforcing the isolation security primitive means considering the underlying software components (the OS, VMM or container manager) as part of the trusted computing base (TCB), i.e., critical part of the software that, when compromised, can jeopardize the security of the entire system. The TCB must therefore be made completely free of vulnerabilities (e.g. bugs, buffer overflows, etc.) and ideally must be formally proven. This is a difficult task with large and complex code. Existing solutions rely on reducing the attack surface by minimizing the code which is part of the TCB (e.g., microkernels like seL4 and unikernels like MirageOS) or by hardening it (e.g., by adding access control, an IDS or a firewall [31]). Today, software solutions offer some form of *isolation* and *peripheral isolation* (discussing to which extent being beyond the scope of this paper). However, assuring strong isolation guarantees in software is still an open issue (see [28] for a recent survey) and side channel attacks remain prominent [17] (thus, the mention 'satisfied with limitations' in Table 3). Moreover, since using software as a root of trust is still an unresolved problem [38], *confidentiality* against the PDMS owner and *attestation* of the code execution cannot be achieved purely through software. As a conclusion, the advantage of using VMMs in terms of *extensibility* is obvious, but such solutions should be considered in combination with other solutions (typically, secure hardware) to achieve the desired *confidentiality* and *attestation* properties of a PDMS (as summarized in Table 3).

Table 3. Claimed properties of different execution environments (left) and required properties of the different modules of our ES-PDMS architecture (right)

Architecture	Claimed properties				
	Code isolation	Attestation	Confidentiality	Peripherals isolation	Extensibility
Secure Element	✓	✓	✓		
TrustZone	✓	✓		✓	≈
Intel SGX	✓	✓	✓		✓
Software	≈			≈	✓

Modules	Required properties				
	Code isolation	Attestation	Confidentiality	Peripherals isolation	Extensibility
Core	✓	✓	✓		
Data collector	✓				✓
Personall computation	✓	✓			✓
Collective computation	✓	✓	✓		✓
Data dissemination	✓			✓	✓
Applications					✓

The properties can be satisfied (✓) or satisfied with limitations (≈)

Hardware-based trusted execution environments (TEE). Hardware TEEs target many different kinds of devices, from personal computers and IoT devices to cloud servers. The most prominent TEEs include secure elements as SIM cards (e.g., in smartphones), ARM TrustZone [13] for SoCs and CPUs integrated into smartphones, tablets and smart appliances, and Intel SGX [25] embedded in all recent Intel CPUs present in personal computers and cloud servers. Although there is still no unique security definition of TEE and their

capabilities vary depending on proposals [45], most TEEs offer capabilities to run code in isolation and remote attestation, which allows it to prove required properties of the code running to third parties. Table 3 summarizes the security tools offered by three technologies: SGX, TrustZone and secure elements (smartcard). These three solutions mainly vary in the way *confidentiality* and *peripheral isolation* are achieved and in terms of the possible level of *extensibility* of the code they can run:

- **Intel SGX** [25] integrates cryptographic primitives in hardware to isolate applications within enclaves, while providing confidentiality and attestation of the code executed in the enclave. Additionally, it provides mechanisms for managing the information flow from code running in the TEE to the outside world, both secure external storage primitives thanks to encryption and management of the communications between processes running in TEEs. It can be used both for securing Cloud apps and in the context of personal computers, leading to an advanced form of extensibility.
- **ARM TrustZone** technology isolates sensitive code executed in the *secure area* of the CPU, from application code executed in the *rich area*. It also aims to *isolate the device's peripherals* (typically, the screen of a smartphone) once code inside the secure area is executed. The code executed inside the secure area has some limitations in terms of resources (e.g., TrustZone ARM1176JZ based on Cortex A series is clocked at 772 MHz and can access several tens of MBs of RAM) leading to a certain form of *extensibility*.
- **Secure elements**, on the other hand, offer much less resources (e.g., advanced secure elements like ST33 based on ARM SecureCore SC300 Cortex M series is clocked at 60 MHz and has only 50KB RAM), thus having a negative impact on extensibility, if used to run a data task. In terms of security, on the contrary, such components provide –in addition to isolation and attestation– confidentiality (with strong guarantees due to tamper-resistance) for running code and data.

The common security primitives needed by the different modules of our ES-PDMS architecture and the primitives provided by these TEEs are reported in Table 3. Several conclusions can be drawn regarding the implementation of the logical architecture we envision: (i) it cannot be implemented using a single technology since none currently ensures all the required properties, and therefore has to combine several elements; (ii) the Core relying on isolation, confidentiality and attestation, can be run on a secure element or on SGX, or be implemented by a combination of execution environments (e.g., pioneer works like TrustVisor [38] propose a secure hypervisor based on combining a secure element with an hypervisor to provide confidentiality and attestation); and (iii) the data tasks needing peripherals isolation (i.e., decision-making) can only be run on TrustZone or a hypervisor/VMM.

An important remark concerns another implicit property in our architecture, which cannot be directly ensured by secure hardware alone. That is the data tasks are exclusively controlled by the Core and the communications between the Core and the data tasks are protected, despite the OS (executing said tasks, e.g., within an SGX enabled CPU) being potentially compromised. In itself neither attestation nor isolation directly provide this property, and some code encapsulation has to be provided in order to make sure that the input/output behavior of the data task cannot be observed by the untrusted OS. A solution for SGX enclaves can be found in [14]. It leverages secure channels and the attestation mechanisms provided by SGX in order to allow for protected execution of an arbitrary remote computation with confidentiality of the input/output behavior.

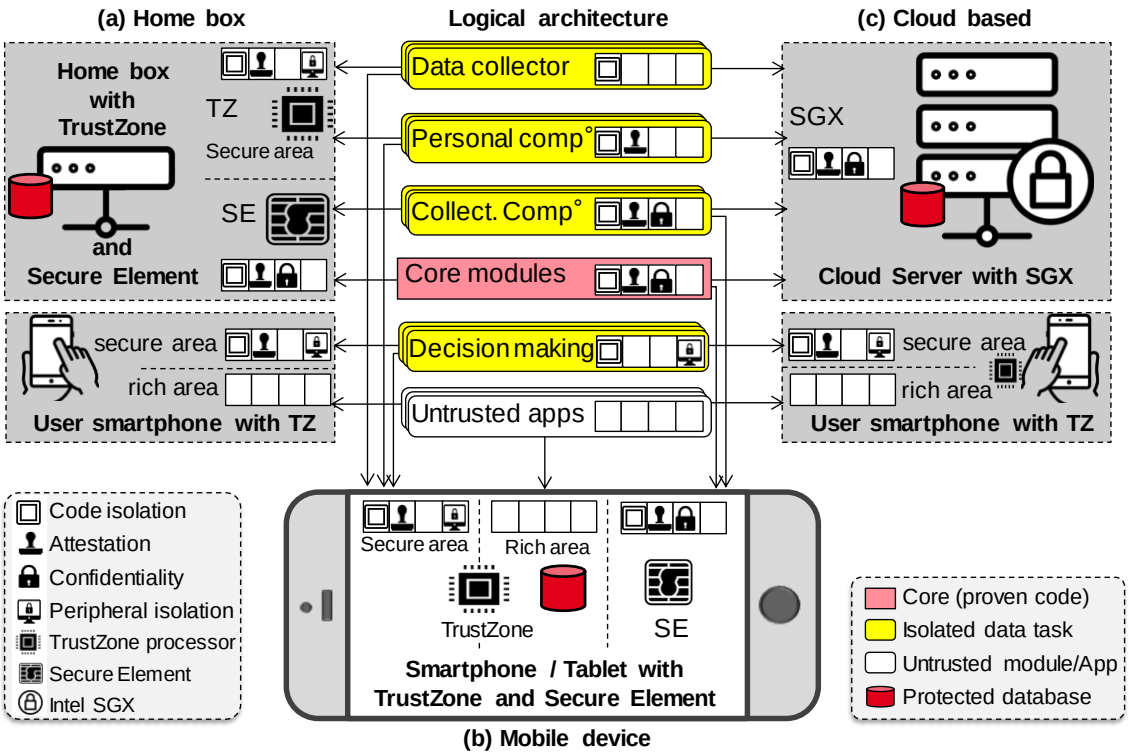

Figure 9. Physical ES-PDMS instances: (a) home box; (b) mobile device; (c) cloud based

The above listed software and hardware solutions can be combined in many ways to achieve the security primitives needed to implement the proposed ES-PDMS logical architecture, but a complete study of the potential architectures, their impact on the data management tasks and their limitations, constitute open challenges. In the following, we limit to presenting a few basic examples of physical architectures encompassing the security properties of an ES-PDMS.

Physical architectures of an ES-PDMS. In Figure 9 we propose three illustrative physical instances of our logical architecture adapted to three different ES-PDMS configurations based on a home box, a personal device and the cloud:

- ***Home box*** (Figure 9.a). The data is stored in the box and is cryptographically protected. The Core runs on a secure element as well as the collective computation tasks[10] to benefit from its security and achieve the attestation and confidentiality requirements. Personal computation and data collector tasks are executed on a TrustZone CPU equipping the box, and thus benefit from extensibility. If no peripheral is available on the box to interact with the user (e.g., no touch screen), the apps and the decision-making tasks run on the smartphone of the PDMS owner, in the rich area and the trusted area respectively, to safely capture the user's I/O.
- ***Mobile device*** (Figure 9.b). Similarly to the home box, the Core and collective computation tasks are operated in the secure element of the smartphone (i.e., an additional SIM card slot is needed), the other tasks (personal computation, data collector and decision-making) run in the trusted area of the TrustZone CPU of the smartphone, while the apps run in the rich area of the CPU.
- ***Cloud*** (Figure 9.c). The SGX-based instance pictured on the right part of Figure 9 is running both the Core and the data tasks in distinct SGX enclaves. The enclave running the Core takes advantage of the attestation capabilities of SGX to control the other enclaves, and collective computations can be performed seamlessly (as well as other personal computation and data collector tasks) given the confidentiality property offered by SGX, with the Core relaying the remote attestation guarantees to the other participants of the protocol. To protect the user's I/O with the decision-making console, this architecture also needs a smartphone with a TEE (e.g., ARM TrustZone).

[10] The collective computation data tasks being operated on a secure element are highly secure (tamper resistant) at the price of extensibility. However, advanced distributed data processing tasks must be done with the support of a more powerful CPU integrated in the box and connected to the secure element, which is an open issue.

Before concluding this section, we describe a preliminary implementation which prefigures the home box architecture instance presented above and constitutes a reasonable base for a first validation of the ES-PDMS concept.

A concrete ES-PDMS instance. A consortium, built around Inria, UVSQ and a French company, has been set up in 2018 with the goal to conceive and deploy a secure decentralized social-medical folder facilitating the coordination of social and medical care at home for elderly people. In a first step, 10 thousand patients from the Yvelines district in France are targeted. In terms of hardware, the solution is close to the home box case presented above. The box is based on a combination of a secure element (SE) hosting the Core and a microcontroller hosting basic data tasks. Applications dedicated to social and medical workers as well as the patient's decision making app run on smartphones. The microcontroller is a STM32F based on an ARM Cortex-M4 CPU with 168KB of RAM and 1MB of embedded NOR where data tasks code resides. The Core itself will run in a ST33J MCU, a certified tamper-resistant SE (Common Criteria EAL6+) equipped with an ARM SC300 CPU, 50KB of RAM and 1 MB of NOR, where the Core code and metadata linked to its secure functioning reside. The porting of the Core on the ST33J is under way, while in the current version of the box a reduced, essential set of security properties is provided by a TPM (Trusted Platform Module). The three main modules of the Core are implemented as follows:

- ***Data storage.*** The data storage module incorporates an embedded database engine based on PlugDB [10, 11, 33] which manages an encrypted database hosting the personal data on a large Flash memory (GBs microSD card) and a recovery procedure based on the protocol described in Section 4.2.2.
- ***Communication.*** The communication module includes an authentication procedure relying on certificates delivered by a PKI and a preliminary implementation of *TLS trusted* to communicate with the remote services (the application of the practitioners running on their smartphones and external servers hosting social-medical personal data).
- ***Policy enforcement.*** The policy enforcement module is based on RBAC policies defining differentiated views of the personal data to the connected user depending on his role (physician, nurse, social worker, etc.), with a reference monitor enabling the PDMS owner to visualize and control all the enacted authorizations [55] from her smartphone.

The choice of a STM32F microcontroller to run the data tasks has been dictated by economic and energy saving constraints. It limits the scope of the box to a reduced set of predefined *Data collectors* and *Personal computations*, with strong assumptions made on isolation (i.e., considered in our specific context of a rather 'closed' and ad-hoc platform). *Collective computations* are not implemented yet. Since collective data tasks cannot be operated inside an MCU which does not protect data confidentiality, they will be operated inside the secure element as simple sequences of SQL queries evaluated by the Core. Although within this architecture, the level of extensibility is still limited and specific security assumptions are made, the limitations are clearly identified and overcoming them is an integral part of the roadmap. More powerful versions of the box are already planned, to support a full range of data tasks with no impact on the global design.

In conclusion, several existing technologies implement today, at least partially, the security properties required by the proposed logical architecture and combining them to obtain appropriate physical instances of an ES-PDMS in different usage contexts is already possible. Regarding the near future, the current trend suggests that the availability and diversity of TEE technologies will increase. New solutions are already envisioned, like heterogeneous multicore platforms [35] in which security/isolation oriented cores (*à la* SGX) would cohabit with other all-purpose cores, allowing for separation of tasks inside the CPU. We expect the modularity of our logical architecture to be of great help in accommodating new physical instances for such upcoming solutions. While this opens to practical ES-PDMS instances, database challenges linked to the underlying data management features and the PDMS context specificities need to be explored. We discuss below some dimensions of the problem, and introduce certain database challenges linked to the architecture in Section 5.

4.5. Discussion

Different practical solutions based on secure hardware can already give rise to some form of ES-PDMS. However, the problem of implementing an ES-PDMS is large and as such there are important related issues which have to be considered as well. We discuss below some dimensions of the problem and limitations of our proposal, not addressed so far for simplicity and conciseness reasons, or because they are rather orthogonal to the architectural dimension considered as the cornerstone of the ES-PDMS introduced in this paper.

A first aspect concerns the security level of a physical instance used to deploy our logical architecture. We discussed above a few examples of instances by taking into account the required set of security primitives (i.e., a

physical instance is valid if it covers all the required properties of the targeted ES-PDMS module) but without a deeper consideration of the security guarantee level associated with the properties. Thinking that a security property can hold in any condition is wishful even for TEEs. Generally, it is wise to think security in terms of an attack's cost/benefit ratio. From this viewpoint, a Cloud based PDMS instance (e.g., running on Intel SGX) could be considered as more exposed to attacks than a home box instance since in the former case the physical platform is shared by many PDMS instances (although each PDMS uses its own dedicated enclaves), while in the latter case the physical platform is dedicated to running a single PDMS. Also, different environments can offer different protection levels for a same security primitive. For example, a software implemented security primitive is generally considered to have a weaker security level than if it were hardware implemented (e.g., TEE), while some TEEs can be considered more secure than others (e.g., smartcard versus ARM TrustZone).

A second important aspect specific to our architecture is to ensure a clean separation between the application level and the data computation level. Indeed, the apps are considered untrusted in our architecture mainly because one cannot trust in general the user computing environment (typically the OS and browser). Therefore, the apps should ideally have access only to the computation results and never to the raw data. Hence, app developers should push as much as possible of the complex app-related data computations into data tasks leaving thus the sensitive task of app policy enforcement to the Core. A few recent works in the domain of web application security are following a similar approach. For example, systems like Amber [22], DIY [40] and πBox [34] consider isolating and sandboxing application subparts to increase security and user's control with untrusted applications. Amber [22] separates web services from data processing by introducing a data-sharing model relying on secure distributed query mechanisms implementing a reduced subset of SQL. Following a similar approach, πBox [34] proposes to split the app into client and cloud sides, establishes restricted communication channels (read-only, write-only, collect statistics) between the two and relies on sandboxes (also coupled with differential privacy) to counter attacks conducted through applications. Our three-tier logical architecture generalizes this separation principle, i.e., application versus data computation, and thus lays the foundation of a framework enabling a number of good security practices for app developers. Besides, we note that the guarantees offered by the above mentioned frameworks are inherently weaker than the guarantees provided by secure hardware. Thus, applications implemented using these frameworks can essentially be considered as weakly secure, based on extensible data tasks satisfying isolation and peripheral isolation (provided the user's web browser is trusted). Also, these frameworks do not provide confidentiality and attestation, two important security properties for an ES-PDMS, which further underlines the importance of secure hardware in this context.

A third aspect is related to data destruction and more generally the usage control offered by the ES-PDMS. In terms of functionalities, we focused in this paper on five main stages of the personal data life-cycle. One might object that data destruction represents also an important stage and should be considered as such. While we agree with this observation, we argue that data destruction raises issues that are mainly outside the scope of the architectural study of this paper. For example, in the case of secondary copies of user data (e.g., email and other social interaction data, health, employment, insurance), the main issue is not deleting the copies of the data collected into the PDMS but destroying the original data managed by the external service that generated it. Thus, the problem shifts to a legal issue, e.g., see the "right to be forgotten" on the Internet in the European legislation. Also, regarding the primary copies of some user data (e.g., quantified-self data, smart home data, photos, videos and documents generated by the user), a general problem appears if the users disseminates the data to other users. In this case, a form of user control can be maintained if the data is shared with another ES-PDMS (e.g., a sticky policy is associated to the shared object and is enforced by the reference monitor module of the PDMS). However, there are major limitations in practice, e.g., how can one preclude a malicious user from making a screenshot when she visualizes a photo that has been shared with her. Hence, data destruction leads to important but very challenging problems such as enforcing usage control or making users aware of their responsibilities.

Many other important problems remain to be addressed such as personal data visualization, data integration, legal, economic or cognitive aspects, or user perception regarding security. Although quite independent of architectural issues, we argue that the extensible architecture introduced in this paper offers interesting means (through data tasks) to tackle such issues without decreasing the security level of the solution. In the following section we discuss a few important challenges from the data management perspective and from an architectural angle.

5. Research challenges towards an ES-PDMS architecture

Without claiming to be exhaustive, we believe that at least three main challenges arise from the core of our analysis of the architectural vision introduced in the paper. The first challenge directly stems from the reference

architecture presented above and concerns its formal analysis. The second challenge relates to the control tools derived from the enforced position and responsibilities acquired by the PDMS owners. The third challenge concerns the concrete implementation and enforcement of distributed data processing without leaks using Trusted Execution Environments. These three challenges are described below.

Challenge 1*: *Design and formal definition of the Core engine. As sketched in Section 4, the Core engine is expected to provide at least authentication and secure personal data storage and access, to enforce security and privacy policies, and to sequence the execution of data tasks installed under the user's control. While existing work (e.g., relational algebra, iterator model and classical query rewriting to access data) can be reused to this end, a particular focus must be put on minimizing the code size and complexity such that the Core engine can be proven through formal methods. This leads to delegate non-critical operations to data tasks potentially based on existing –non proven and large– code modules or libraries, interacting with the Core without jeopardizing the global security of the computations.

A modular architecture such as the one we propose allows to construct independent definitions of the security goals, rigorous proofs for each Core component, and rely solely on the security of the Core and on the security properties of the data tasks (rather than on their implementation). While such an approach is common in cryptographic protocol design, the complexity lies in the large amounts of data and large scale of the system considered here, compared to usual cryptographic protocols. The modularity of our architecture can help in solving these issues, as designing provably secure components allows abstracting away lower level details. A good example of such a modular design is the secure map-reduce framework VC3 [48] which builds on top of SGX's local isolation and attestation guarantees to prove security and integrity of the whole computation. A similar approach could be adapted to the combination of components described in this paper. Solutions already exist to model the low-level properties of data tasks as described in Section 4.1 (e.g., [14, 16]), which could form a solid foundation for higher-level properties.

Challenge 2: Controlled data sharing. Defining a well-calibrated data sharing policy, and enforcing it despite piracy actions, requires an expertise which is out of reach of most individuals. Of course, new data sharing and usage control techniques dedicated to personal cloud lay users should be invented. Although this issue is general and might go beyond the strict case of the PDMS context [54], a new challenge specific to the PDMS context is the one of making the PDMS owner able to define her own sharing policy without being forced to understand the underlying access control semantics. To this end, the administration of data sharing should rely on three properties: (i) *visualization*: whatever the underlying access control model, the owner should have the capacity to visualize the net effects of these rules and to easily adjust them if required; (ii) *trusted reference monitor*: the reference monitor logic enforcing the sharing policy must itself be understandable by the holder and the platform implementing this logic must be trusted by her; and (iii) *zero-knowledge grants*: side channels should be proscribed to avoid leaking personal information within authorized sets of permission. Each of these properties leads to specific challenges.

Visualization. Regarding the first point, existing works like πBox [34] promote the idea of letting users visualize the effects of their decisions before validating any sharing action. In the personal cloud context however, the amount of permissions can be huge. A first issue is thus to assist the holder in this validation task. A recent study [55] makes one step in this direction by identifying suspicious grants, thanks to 'watchdog triggers' comparing new authorizations with previous ones to identify outliers. A second issue is to help the owner regulate, and then visualize, the complete lifecycle of her personal data, e.g., from their capture to their dissemination, all the way to their deletion while the data may undergo transformations (e.g., aggregation, anonymization) at some steps of this lifecycle to reduce its sensitivity. Moreover, the effects of some rules are challenging to visualize (e.g., time or location-based contextual rules). Hence significant work remains to be done to define such friendly visualization tools. The challenge is important since providing a visual feedback to the individuals about the way they are exposed greatly helps them to adapt their behavior [5, 30].

Trusted reference monitor. As shown in Section 4, visualization tools only make sense if the holder can trust what is actually plotted. This means designing visualization tools that can be integrated in data tasks providing peripherals isolation, and provide to the PDMS owner means to trust the reference monitor to precisely enforce the effect visualized. Therefore, the reference monitor logic must be basic to be easily understood by the holder and must be integrated in the Core and run inside a TEE. A primary solution is to materialize all permissions by

means of ACLs (i.e., user u is granted access a to resource r if $(u, r, a) \in$ ACL list), whatever the complexity of the underlying access control model producing these ACLs. This leads to a materialized view maintenance problem in the TEE and to an embedded data management problem to minimize the reference monitor overhead at data access time.

Zero-knowledge grants. A third issue is to proscribe by design any side channel to be created between the personal cloud and a remote party. For example, a set $\{(u, r, a)\}$ of authorizations could be consider acceptable by the PDMS owner when each triple (u, r, a) is examined individually, but may reveal unexpected sensitive information from the PDMS through a side channel when considered globally. For example, imagine a sharing model extracting from *Alice*'s PDMS a set of ACLs authorizing her friend *Bob* to access to certain photos. The set of photos authorized to a *Bob* could be legitimate when considered individually (i.e., a legitimate permission, granted to a legitimate subject, on a regular object), but could globally leak information about the content of *Alice*'s PDMS (e.g., the photos being ordered on size, the values of the ith pixel of each picture may be used as a side channel and leak the credit card number of *Alice*). Building on the proposed architecture, specific system data tasks could be imagined to circumvent such issues. Typically, verification techniques based on the replay of the data tasks checking the resulting permissions on a 'what if' basis (e.g., replay the sharing rule on a dataset of photos with and without the banking information) may help reveling such side channels. Of course, this problem is not limited to the PDMS context, but is of great importance here as data sharing decisions are made by lay users on the basis of their entire digital assets.

Challenge 3: Secure distributed data-oriented computations using trusted execution environments. The personal cloud paradigm allows for novel big data applications on personal data (e.g., participatory sensing, epidemiological studies, personalized recommender systems, etc.). More precisely, the objective of distributed database computations in the PDMS context is being able to compute any function taking as inputs the private values of n PDMS owners without leaking any information other than the final result. This is close to the well-known secure multiparty computation (MPC) problem. While secure distributed –personal– data processing is not a new issue, the architecture promoted in this paper introduces new specificities which open up to new possibilities but also raises new challenges. First, the PDMS architecture is distributed at the individual level and is expected to scale up to nationwide populations. While MPC protocols have been widely studied in cryptography, they have not been designed with scalability in mind and large-scale solutions exist only for a very limited set of functions [46]. Second, the security of the architecture we propose relies on secure hardware (or more precisely, Trusted Execution Environment as described in Section 4.4), which exhibit specific constraints with a potential impact on data management structures and algorithms, and calls for new threat models when compared to traditional MPC. Secure and efficient data processing in this context leads to specific challenges including the following:

Integrity and confidentiality of generic hardware-based distributed computations. Relying on the properties of the proposed architecture inherited from a TEE opens up the field for generic and scalable solutions, but has major differences with traditional MPC. First, the adversary model is different. Traditional MPC considers an *honest-but-curious* adversary model, which makes sense when the risk of being blacklisted is a strong enough deterrent to force the participating entities (e.g., central servers) to not deviate from the protocol. Under the assumption that the data tasks involved in the distributed computation are isolated thanks to TEEs, even in the presence of a curious or malicious PDMS owner, the data task sticks close to a *fully honest* behavior. A first research issue is thus to propose a way to map any arbitrary distributed protocol within an understandable manifest specifying the overall orchestration of the computation (i.e., specifying the role and privileges of each data tasks involved), and ensure at execution that the computation protocol is respected as stated in a manifest. This is not an easy task for several reasons. First, the secure execution of the distributed computation protocol may be enforced on a *local correctness checking* basis, i.e., each participant involved in the computation should be given means to check the correctness of the distributed protocol from its local view and be guaranteed that, if the local check is verified, the global output is also correct. Second, in the case of many data oriented operations (e.g., hashing or sorting as part of joins, group by or duplicate removal) the data flow specified between the data tasks is not deterministic and independent of the data content, but can lead to transferring data to a subsequent data task depending on the data value (e.g., *Alice*'s data tasks transmits data to *Bob*'s data task according to a hash of the data value) which may reveal personal information to an attacker able to observe the data exchanges. *Counter measures* should be thus envisioned to enrich the manifest, e.g., with pre-processing data sampling steps or introduction of dummy data exchanges into the manifest. Third, while TEEs are supposed to guarantee the confidentiality of the data task content, this security property cannot be considered in practice as unconditionally unbreakable. Indeed, even if secure hardware is used,

a fraction of the participating PDMS owners may have instrumented their PDMS platform and exploit side channel attacks to retrieve the content of the data tasks they are in charge of. Such attacks should be taken into account when building the computation manifest, to avoid letting an attacker compromise a large set of users' privacy or being able to target a specific user. A new expected property should thus be introduced, called *data locality*, specifying that any information in possession of any involved *data task* should be as close as possible to the personal data contributed by the PDMS owner of that data task. This property is essential to reason about corruption, define relevant metrics quantifying the potential impact on data leakage of colluding participants with instrumented hardware, and design and integrate appropriate counter measures in the manifest to minimize the benefit-to-cost ratio of such attacks.

Performance of large scale data-oriented operations under the TEE constraints. Considering the PDMS architecture presented in Section 4, an important challenge is obviously to investigate the use of Trusted Execution Environments (TEEs) available at user side (e.g., Intel SGX available in PCs, ARM TrustZone in smartphones/tablets and home boxes). In the database area, there is an intense research agenda on data management for new and/or specialized hardware [6, 35] and systems like Oracle M7 even start to provide hardware implementations of rich sets of database operations (SQL in silicon). But the focus is on secure centralized databases or on the database-as-a-service (DaaS) context. Typically, TrustedDB [15] considers a database running of a tamper proof dedicated secure hardware, Cypherbase [12] adjuncts a secure hardware to an SQL-Server database and EnclaveDB [42] considers a transactional database running inside an SGX enclave. The typical limitations of the cryptographic overhead of accessing persistent data outside the TEE enclave [20], the limited RAM amount of each TEE enclave [25], the cost of external function calls [59] and memory access overheads [18], may slow the computing by orders of magnitude compared to a regular environment, and have to be taken into account. Preliminary works in this direction try to take advantage of several Intel SGX enclaves [25, 48] to parallelize the processing in a secure manner. The problem mentioned above of enforcing the *local correctness checking* and *leakage resilience* properties hence turns the decentralized challenge itself into a performance issue. A precise analysis of TEEs enclaves w.r.t. data-oriented operations in a distributed context still remains to be done and will undoubtedly raise many interesting data management challenges.

6. Conclusion

The personal cloud paradigm is coming at a rapid pace. The goal is to empower individuals with their personal data and to unlock new usages founded on the use of their personal data under their control. The recent worldwide smart disclosure initiatives and privacy regulations offer great support for the personal cloud movement. However, the journey is not without danger and many solutions have taken a path that eventually leads to empower users in a way which may exacerbate privacy risks.

A first approach was to consider that the personal cloud context could be addressed with a simple adaptation of classical corporate cloud techniques, hence the proliferation of online personal cloud services. This is unfortunately not the case. While corporate solutions address a carefully controlled set of applications and are tuned towards data management and security experts, the PDMS context sketches an open and rich ecosystem of untrusted data processing apps in interaction with an unsecure execution environment and a layman PDMS owner. Trying to answer the fear associated with online personal clouds, zero-knowledge and home cloud solutions emerged but in fact, they transformed the trust assumption in the personal cloud provider into the myth of the owner's self-capacity to administer and guarantee the security of her own environment.

Solving this issue is both a scientific, technical and societal challenge. On the scientific side, the primary challenge addressed to the data management and security research communities is to define and formally analyze a personal cloud architecture capable of combining strong expectations in terms of security with extensible data management capabilities covering the complete personal data life cycle. Our definition of an Extensive and Secure Personal Data Management System (ES-PDMS) along with a set of security properties and reference architecture is a first step towards this goal but many exciting research issues still need to be investigated. On the technical side, we have sketched different alternatives for instantiating this reference architecture into concrete platforms but a long road remains before achieving operational solutions exploiting the full capacity of advances in secure hardware. Some steps along this road are undoubtedly difficult enough to introduce new research issues on their own. Finally, the societal challenge is multidisciplinary: how to devise new economic models reconciling lucrative (or non-profit) exploitation of personal data and user's empowerment? How to make the legislation evolve in relation with the evolution of PDMS architectures, notably regarding the sharing of liability between the PDMS provider, the application providers and the owner herself? And finally, how to convince people of the importance

of all the previous questions and provide them with the right choices? Architecture also matters when devising solutions to these questions. It must provide formal guarantees that the interest of each party will be preserved, that each party will get the appropriate tools to endorse her own responsibility and finally it must be as transparent and friendly to use as possible.

Hence, we believe that the personal cloud paradigm is a cornerstone of the digital evolution and that many of its challenges are rooted in architectural choices. Our hope is that this paper will provide a solid foundation for the discussions around this promising topic and lead to impactful and original research.

Acknowledgements

This research is supported by the Inria Innovation Lab 'OwnCare' and by the ANR PersoCloud grant.

References

[1] Abiteboul, S., André, B. and Kaplan, D. Managing your digital life. CACM, 58, 5 (2015).

[2] Adi Shamir: How to Share a Secret. Commun. ACM 22(11), 1979.

[3] Allard, T., Anciaux, N., Bouganim, L., Guo, Y., Le Folgoc, L., Nguyen, B., Pucheral, P., Ray, I., Ray, I. and Yin, S. Secure personal data servers: a vision paper. VLDB, 2010.

[4] Allard, T., Hébrail, G., Masseglia, F., Pacitti, E. Chiaroscuro: Transparency and privacy for massive personal time-series clustering. SIGMOD, 2015.

[5] Almuhimedi H., Schaub F., Sadeh N., Adjerid I., Acquisti A., Gluck J., Cranor L. F., Agarwal Y. Your Location has been Shared 5.398 Times!: A Field Study on Mobile App Privacy Nudging. CHI, 2015.

[6] Alonso, G. Data Processing in Modern Hardware. Tutorial at EDBT, 2016.

[7] Amiri Sani, A. SchrodinText: Strong Protection of Sensitive Textual Content of Mobile Applications. ACM International Conference on Mobile Systems, Applications and Services (MobiSys), 2017.

[8] Anciaux, N., Bonnet, P., Bouganim, L., Nguyen, B., Popa, I. S. and Pucheral, P. Trusted Cells: A Sea Change for Personal Data Services. CIDR, 2013.

[9] Anciaux, N., Bouganim, L., Pucheral, P., Guo, Y., Le Folgoc, L. and Yin, S. (2014). MILo-DB: a personal, secure and portable database machine. Distributed and Parallel Databases, 32(1).

[10] Anciaux, N., Bouganim, L., Pucheral, P., Guo, Y., Le Folgoc, L., and Yin, S. MILo-DB: a personal, secure and portable database machine. DAPD, 32, 1 (2014).

[11] Anciaux, N., Lallali, S., Popa, I. S., Pucheral, P. A scalable search engine for mass storage smart objects. PVLDB, 2015.

[12] Arasu, A., Eguro, K., Joglekar, M., Kaushik, R., Kossmann, D. and Ramamurthy, R. Transaction processing on confidential data using cipherbase. ICDE, 2015.

[13] ARM. Security Technology-Building a Secure System using TrustZone Technology. ARM Technical White Paper, 2009.

[14] Bahmani, R., Barbosa, M., Brasser, F., Portela, B., Sadeghi, A. R., Scerri, G., and Warinschi, B. Secure Multiparty Computation from SGX. IACR Cryptology ePrint Ar., 2016.

[15] Bajaj, S. and Sion, R. TrustedDB: A trusted hardware-based database with privacy and data confidentiality. Transactions on Knowledge and Data Engineering, 26, 3 (2014).

[16] Barbosa, M., Portela, B., Scerri, G., and Warinschi, B. Foundations of hardware-based attested computation and application to SGX. EuroS&P, 2016.

[17] Bazm, M. M., Lacoste, M., Südholt, M., Menaud, J. M. Side Channels in the Cloud: Isolation Challenges, Attacks, and Countermeasures. Preprint, 2017.

[18] Bilal, M. Towards Secure Stream Processing Using Intel SGX. Report Univ. Louvain, 2017.

[19] Boutet, A., Frey, D., Guerraoui, R., Jégou, A., and Kermarrec, A. M. Privacy preserving distributed collaborative filtering. Computing, 98, 8 (2016).

[20] Brenner, Stefan, et al. SecureKeeper: Confidential ZooKeeper using Intel SGX. Middleware, 2016.

[21] Brickell, E., Camenisch, J. and Chen, L. Direct anonymous attestation. ACM International Conference on Computer and Communications Security (CCS), 2004.

[22] Chajed, T., Gjengset, J., Van Den Hooff, J., Kaashoek, M. F., Mickens, J., Morris, R. and Zeldovich, N. Amber: Decoupling user data from web applications. 15th Workshop on Hot Topics in Operating Systems (HotOS XV), 2015.

[23] Chaudhry, A., Crowcroft, J., Howard, H., Madhavapeddy, A., Mortier R., Haddadi, and H. McAuley, D. Personal data: thinking inside the box. Aarhus Conference on Critical Alternatives, 2015.

[24] Christl, W., Kopp, K. and Riechert, P. U. Corporate surveillance in everyday life. Cracked Labs, 2017.

[25] Costan, V. and Devadas, S. Intel SGX Explained. IACR Cryptology ePrint Archive, 2016.

[26] Dalskov, A. P. and Orlandi, C. Can You Trust Your Encrypted Cloud?: An Assessment of SpiderOakONE's Security. ACM Asia Conference on Computer and Communications Security (AsiaCCS), 2018.

[27] deMontjoye, Y. A., Shmueli, E., Wang, S. S. and Pentland, A. S.. OpenPDS: Protecting the privacy of metadata through safe answers. PloS one, 9, 7 (2014).

[28] Fernandes, D. A., Soares, L. F., Gomes, J. V., Freire, M. M., Inácio, P. R. Security issues in cloud environments: a survey. Information Security, 13, 2 (2014).

[29] French Law on Statistical Confidentiality. Loi n° 51-711 du 7 juin 1951 sur l'obligation, la coordination et le secret en matière de statistiques.

[30] Harkous, H., Rahman, R., Karlas, B. and Aberer, K. The Curious Case of the PDF Converter that Likes Mozart: Dissecting and Mitigating the Privacy Risk of Personal Cloud Apps. arXiv preprint, 2016.

[31] Ibrahim, A. S., Hamlyn-Harris, J., and Grundy, J. Emerging security challenges of cloud virtual infrastructure. arXiv preprint, arXiv:1612-09059, 2016.

[32] Katz, J. Universally composable multi-party computation using tamper-proof hardware. International Conference on the Theory and Applications of Cryptographic Techniques, 2007.

[33] Lallali, S., Anciaux, N., Popa, I. S., and Pucheral, P. Supporting Secure Keyword Search in the Personal Cloud. Information Systems, 72 (2017).

[34] Lee, S., Wong, E. L., Goel, D., Dahlin, M., Shmatikov, V. ?box: A platform for privacy-preserving apps. NSDI, 2013.

[35] Lehner, W. The data center under your desk: how disruptive is modern hardware for DB system design? Keynote at PVLDB, 10, 12 (2017).

[36] Lentz, M., Sen, R., Druschel, P. and Bhattacharjee, B. SeCloak. ARM Trustzone-based Mobile Peripheral Control. ACM International Conference on Mobile Systems, Applications and Services (MobiSys), 2018.

[37] Luu, T., Klemm, F., Podnar, I., Rajman, M. and Aberer, K. Alvis peers: a scalable full-text peer-to-peer retrieval engine. Int. Workshop on Inf. Retrieval in P2P Networks, 2006.

[38] McCune, J. M., Li, Y., Qu, N., Zhou, Z., Datta, A., Gligor, V., Perrig, A. TrustVisor: Efficient TCB reduction and attestation. S&P, 2010.

[39] Mortier, R., Haddadi, H., Henderson, T., McAuley, D. and Crowcroft, J. Human-data interaction: the human face of the data-driven society. Available at SSRN 2508051, 2014.

[40] Palkar, S. and Zaharia, M. DIY Hosting for Online Privacy. In ACM Workshop on Hot Topics in Networks, 2017.

[41] Popa, R. A., Redfield, C., Zeldovich, N., and Balakrishnan, H. CryptDB: protecting confidentiality with encrypted query processing. ACM Symposium on Operating Systems Principles (SOSP), 2011.

[42] Priebe, C., Vaswani, K. and Costa, M. EnclaveDB: A Secure Database using SGX. IEEE Symposium on Security & Privacy (S&P), 2018.

[43] Ratnasamy, S., Francis, P., Handley, M., Karp, R. and Shenker, S. A scalable content-addressable network. ACM 31, 4 (2001).

[44] Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation, GDPR).

[45] Sabt, M., Achemlal, M., and Bouabdallah, A. Trusted execution environment: What it is, and what it is not. In Trustcom/BigDataSE/ISPA, 2015.

[46] Saia, J. and Zamani, M. Recent results in scalable multi-party computation. Int. Conf. on Current Trends in Theory and Practice of Informatics, 2015.

[47] Schmidt, C. and Parashar, M. Squid: Enabling search in DHT-based systems. Journal of Parallel and Distributed Computing, 68, 7 (2008).

[48] Schuster, F., Costa, M., Fournet, C., Gkantsidis, C., Peinado, M., Mainar-Ruiz, G. and Russinovich, M. VC3: trustworthy data analytics in the cloud using SGX. S&P, 2015.

[49] STMicroelectronics. ST33Jxxx. STM data brief 028629 Revision 3, February 2017.

[50] Stoica, I., Morris, R., Karger, D., Kaashoek, M. F. and Balakrishnan, H. Chord: A scalable peer-to-peer lookup service for internet applications. SIGCOMM Computer Communication Review, 31, 4 (2001).

[51] Task Force on Smart Disclosure. Smart Disclosure and Consumer Decision Making. Report, NST Council, 2013.

[52] That, D. H. T., Popa, I. S., Zeitouni, K. and Borcea, C. PAMPAS: Privacy-Aware Mobile Participatory Sensing Using Secure Probes. ACM International Conference on Scientific and Statistical Database Management (SSDBM), 2016.

[53] To, C. Q., Nguyen, B. and Pucheral, P. Private and Scalable Execution of SQL Aggregates on a Secure Decentralized Architecture. TODS, 41, 3 (2016).

[54] Tran-Van, P., Anciaux, N., and Pucheral, P. A New Sharing Paradigm for the Personal Cloud. Trustbus, 2017.

[55] Tran-Van, P., Anciaux, N., and Pucheral, P. SWYSWYK: a Privacy-by-Design Paradigm for Personal Information Management Systems. ISD, 2017.

[56] Wang, J. and Wang, Z. A survey on personal data cloud. The Scientific World Journal, 2014.

[57] Xiaokui Xiao and Yufei Tao. Personalized privacy preservation. SIGMOD, 2006.

[58] Zhang, L., Litton, J., Cangialosi, F., Benson, T., Levin, D. and Mislove, A. Picocenter: Supporting long-lived, mostly-idle applications in cloud environments. ACM European Conference on Computer Systems (EuroCCS), 2016.

[59] Zhao, ChongChong, et al. On the Performance of Intel SGX. IEEE Web Inf. Systems and Applications Conf., 2016.