Secure Personal Data Servers: a Vision Paper

Tristan Allard[*,**], Nicolas Anciaux[*], Luc Bouganim[*], Yanli Guo[*], Lionel Le Folgoc[*], Benjamin Nguyen[*,**], Philippe Pucheral[*,**], Indrajit Ray[***], Indrakshi Ray[***], Shaoyi Yin[*]

* INRIA Rocquencourt
Le Chesnay, France
<Fname.Lname>@inria.fr

** PRISM Laboratory
Univ. of Versailles, France
<Fname.Lname>@prism.uvsq.fr

*** Colorado State University
Fort Collins, CO, USA
{indrajit,iray}@cs.colostate.edu

ABSTRACT

An increasing amount of personal data is automatically gathered and stored on servers by administrations, hospitals, insurance companies, etc. Citizen themselves often count on internet companies to store their data and make them reliable and highly available through the internet. However, these benefits must be weighed against privacy risks incurred by centralization. This paper suggests a radically different way of considering the management of personal data. It builds upon the emergence of new portable and secure devices combining the security of smart cards and the storage capacity of NAND Flash chips. By embedding a full-fledged Personal Data Server in such devices, user control of how her sensitive data is shared by others (by whom, for how long, according to which rule, for which purpose) can be fully reestablished and convincingly enforced. To give sense to this vision, Personal Data Servers must be able to interoperate with external servers and must provide traditional database services like durability, availability, query facilities, transactions. This paper proposes an initial design for the Personal Data Server approach, identifies the main technical challenges associated with it and sketches preliminary solutions. We expect that this paper will open exciting perspectives for future database research.

1. INTRODUCTION

The number of information systems continuously gathering personal data on servers is escalating at a tremendous pace. Electronic Health Record systems used today in most advanced countries, vehicle tracking systems used to compute insurance premium, and soon carbon tax, travelers tracking systems used by public transportation companies, systems implementing e-administration procedures (scholarship folders, identity cards, social security covers, pension funds, income taxes, …) are illustrative but not exclusive examples. Citizens have no way to opt-out of these applications because governments, public agencies or companies that regulate our daily life require them.

In the meantime, administrations and companies deliver an increasing amount of digitized personal data to the user (salary forms, insurance forms, invoices, phone call sheets, banking statements, etc). Primary copies of this data are kept by the information system that produced the data and secondary copies are delivered to the user for her personal use. While nothing dictates this, these secondary copies often also end up in servers for user's convenience. Indeed, the user expects her data to be resilient to failure, available through the internet 24/7 and easily manageable (i.e., organized, queryable, sharable). Many internet companies provide precisely this service to everyone, sometimes for free, and without requiring any computer expertise from the end user.

All these situations put together result in accumulating a complete digital history of citizens in servers. The benefits of centralizing personal data are unquestionable in terms of data completeness, failure resiliency, high availability and even consistency of security policies. But these benefits must be weighed carefully against the privacy risks of centralization. There are many examples of privacy violations arising from negligence, abusive use, internal attacks, external attacks, and even the most secured servers are not spared. Annex B gives recent examples of large scale privacy violations.

This paper draws a radically different vision of the management of personal data. This vision builds upon the emergence of new hardware devices called Secure Portable Tokens (SPT for short). Whatever their form factor (SIM card, secure USB stick, wireless secure dongle), SPTs combine the tamper resistance of smart card microcontrollers with the storage capacity of NAND Flash chips. This unprecedented conjunction of portability, secure processing and Gigabytes-sized storage holds the promise of a real breakthrough in the secure management of personal data. The idea promoted in this paper is to embed in such devices software components capable of acquiring, storing and managing personal data. However, our approach does not amount to a simple secure repository of personal documents. The ambition is, first, to allow the development of new, powerful, user-centric applications and to serve data requests from existing server-based applications managing personal data, thus requiring a well organized, structured, consistent and queryable representation of these documents[1]. Second, we want to provide the user with a friendly control over the sharing conditions related to her data and with tangible guarantees about the enforcement of these conditions. These two objectives lead to the definition of a real secure and portable Personal Data Server (PDS for short)[2]. With appropriate infrastructure, PDSs enable the vision depicted by Figure 1. Bob's personal data, delivered by different sources, is sent to his PDS which can then serve data requests from private applications (serving Bob's interest), secure multi-actors applications (accessed through actors' PDS) and external applications. Bob's PDS can also take part in secure global processing.

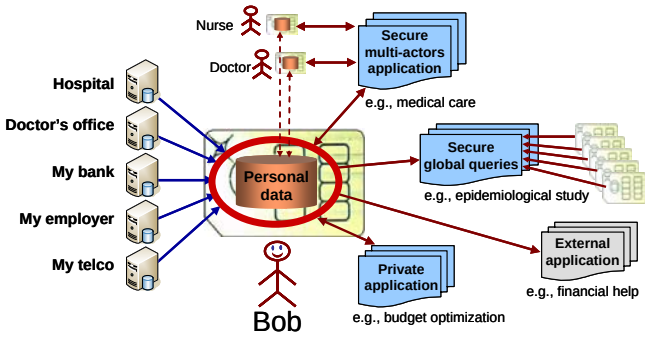


Figure 1. The Personal Data Server approach

[1] Simple document repositories, even if they integrate keyword search facilities cannot meet these requirements. The lack of data structure has been considered as one of the major reasons (with security) that explain the failure of the first French national EHR System [15].

[2] PDS has no connection with the Personal Server notion of [23].

What can be precisely expected from a PDS is the following:

- To provide the main functionalities of a database engine, namely data description and structuring (schema management), access control management, query facilities, and transactional properties.
- To be interoperable (1) with existing data sources to allow the acquisition (and rendering) of personal data, and (2) with other PDSs to allow secure data sharing protocols among them.
- To reestablish the control of the user on how her personal data is shared with others (what data, with whom, for how long, for which purpose). In other words, a PDS must give the ability to enforce privacy principles (e.g., consent, limited collection, limited retention, audit) [3] for all data it stores and for all data it accesses from other PDSs.
- To inherit the portability and tamper-resistance of the device embedding it, thereby providing disconnected facilities and an enforcement of security rules stronger, yet more flexible, than those of a traditional server.

Converting the PDS vision into reality introduces important challenges. First, the SPT, central element of the approach, exhibits strong hardware constraints. Traditional core database techniques (storage and indexing, query and transaction processing) need to be fully revisited to design an embedded database engine that provides acceptable performance. Second, to enforce security rules in a PDS-based information system, atypical distributed protocols combining a large number of highly secure but low power PDSs with a powerful but unsecured server infrastructure must be devised. Third, the PDS approach aims at helping every individual to better protect her privacy. The way to control how data is shared and protected must therefore be highly intuitive and simple. Our hope is that this paper will open various and exciting research directions for the database community.

The rest of the paper is organized as follows. Section 2 presents the characteristics of a SPT and derives from it the problem statement associated with the implementation of the PDS vision. Section 3 illustrates the PDS vision through different scenarios. We sketch out an initial design for the PDS architecture in Section 4, present a set of technical challenges in Sections 5 to 7 and provide concluding remarks in Section 8. Additional material is provided in Annex (preliminary prototypes prefiguring the PDS vision and protocols) to further convince the reader that the PDS vision is not pure utopia.

2. PROBLEM STATEMENT

We introduce the hardware characteristics of SPT, then present the hypothesis related to the security of PDSs and of the infrastructure surrounding them and finally state the problem related to the implementation of the PDS approach.

Hardware characteristics of Secure Portable Tokens: SPTs appear today in a wide variety of form factors ranging from SIM cards to various forms of pluggable secure tokens. Whatever the form factor, SPTs share several hardware commonalities. Their microcontroller is typically equipped with a 32 bit RISC processor (clocked at about 50 MHz today), memory modules composed of ROM, static RAM (about 64KB), a small internal stable storage (about 1MB of NOR Flash) and security modules providing the tamper-resistance. The microcontroller is connected by a bus to a large external mass storage (Gigabytes of NAND Flash). However, this mass storage does not benefit from the microcontroller tamper resistance. SPTs can communicate with the outside world through various standards (e.g., USB2.0, Bluetooth, 802.11). Figure 2 shows typical examples of SPTs but this paper makes no assumption on the form factor.

Hardware progresses are fairly slow in the secure chip domain because the size of the market (billions of units), and the requirement for high tamper-resistance leads to adopting cheap and proven technologies [13]. Nonetheless, SPT manufacturers forecast a regular increase of the CPU power, stable storage capacity and the support of high communication throughputs (up to 480 Mb/s). RAM will unfortunately remain a scarce resource in the foreseeable future due to its poor density. Indeed, the smaller the silicon die, the more difficult it is to snoop or tamper with its processing, but RAM competes with CPU, ROM and NOR in the same silicon die.

In summary, a SPT can be seen as a low power but very cheap (a few dollars), highly portable, highly secure computer with reasonable storage capacity for personal use.

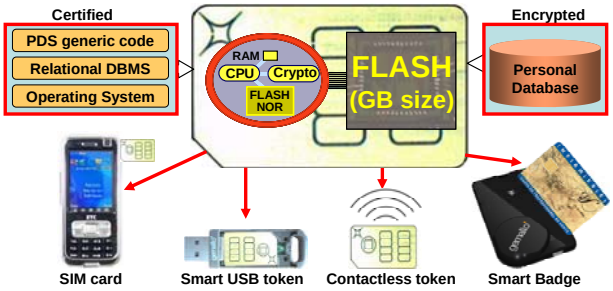


Figure 2: Secure Portable Token and embedded PDS

Hypothesis on the PDS security: the level of trust which can be put in the PDS comes from the following factors:

1. The PDS software inherits the tamper resistance of the SPT making hardware and side-channel attacks highly difficult.
2. The basic software (operating system, database engine and PDS generic tools), called hereafter *PDS core*, can be certified according to the Common Criteria, making software attacks also highly difficult.
3. The PDS core can be made auto-administered thanks to its simplicity, in contrast to its traditional multi-user server counterpart. Hence, DBA attacks are also precluded.
4. Compared to a traditional server, the ratio Cost/Benefit of an attack is increased by observations 1 and 2 and by the fact that a successful attack compromises only the data of a single individual.
5. Even the PDS holder cannot directly access the data stored locally. After authentication (e.g., by a pin code), she only gets the data according to her privileges.

Unfortunately, a PDS cannot provide all the required database functionalities (e.g., durability, if the PDS is lost or destroyed, availability when the PDS is disconnected, global queries involving data from several PDSs) without resorting to external servers, called hereafter *Supporting Servers*.

Hypothesis on Supporting Servers: we assume that Supporting Servers are *Honest* but *Curious*, a common assumption regarding Storage Service Providers. This means that they correctly provide the services that are expected from them (typically serve store, retrieve, and delete data requests) but they may try to breach confidentiality of any data that is stored locally.

Therefore, implementing the PDS approach requires solving the problem stated below:

- To implement the PDS core tackling the SPT hardware constraints.
- To implement the traditional functions of a central server (durability, availability, global queries) in a secure way using Honest but Curious Supporting Servers.

- To provide the user with intuitive tools and underlying mechanisms helping her to control how her personal data is shared.

The next two sections assume that the problem stated above can be solved. Then Section 5, 6 and 7 discuss the technical challenges associated to each dimension of this problem.

3. MOTIVATING EXAMPLES

3.1 Healthcare scenario

Alice carries her electronic healthcare folder (along with other information) on a PDS. She has an account on *e-Store*, a Supporting Server provider. She downloaded in her PDS, from the Ministry of Health, a predefined healthcare database schema, an application to exploit it, and an access control policy defining the privileges attached to each role (physician, nurse, etc). Alice can manage the authorizations by herself, for example by selecting which physicians can play this role. When she visits Bob, a new physician, she is free to provide her SPT or not, depending on her willingness to let Bob physically access it (this is a rough but effective way to control the sharing of her data, as with a paper-based folder). In the positive case, Bob plugs Alice's PDS on his terminal, authenticates to the PDS server with his physician credentials, queries and updates Alice's folder through his local Web browser, according to the physician's privileges.

Bob prescribes a blood test to Alice. The document containing the blood test results is sent to Alice by the medical lab in an encrypted form, through e-Store acting here as a secure mailbox. The document is downloaded from e-Store and wrapped by Alice's PDS to feed the embedded database. If this document contains information Alice would like to keep secret, she simply masks this document so that it remains hidden from any user querying the database except her. The lab keeps track of this medical act for administrative purposes but does not need to keep a copy of its medical content. If Alice loses her PDS, its tamper-resistance renders potential attacks harmless. She may then recover her folder from an encrypted archive stored by e-Store.

Alice suffers from a long-term sickness and must receive care at home. Any practitioner can interact at home with Alice's PDS thanks to his netbook, tablet PC or PDA without need for an Internet connection. To improve care coordination, Bob convinces Alice to make part of her folder available 24/7, during a one month period, to him and to Mary, a specialist physician. A1ice uploads the requested part of her folder encrypted on e-Store. The secret key is exchanged with Bob's and Mary's PDSs in order for them to be able to download Alice's data on their own local database and query it. While Alice's data is now replicated on Bob's and Mary's PDSs, Bob and Mary cannot perform actions on the replica exceeding their privileges and this replica will be destroyed after a one month period because their PDS will enforce these controls. Bob and Mary's actions are recorded by their own PDSs and sent back to Alice through e-Store for audit purpose. To make this sharing scenario possible, patients and practitioners are all assumed to be equipped with PDSs and these PDSs are assumed to share a compliant database schema. As shown in Annex A, which presents a field experimentation, this assumption is realistic in several practical situations.

Finally, if the Ministry of Health decides to compute statistics or to build an anonymized dataset from a cohort of patients, the targeted PDSs will perform the processing and deliver the final result while preventing any leakage of sensitive data or identifying information.

3.2 Vehicle tracking scenario

John, a traveling salesman, drives a car from his company during working hours and uses his personal car otherwise that he shares with Cathy, his daughter who has just got her driving license. Both have a PDS that they plug in the car to register all their personal trips. Several applications are interested in the registered GPS locations. John's insurance company adapts the insurance fee according to different criteria (e.g., the distance traveled, type of road used, and speed). Cathy will probably pay more than her father because she lacks enough driving experience. The Treasury is also interested by this information to compute John's carbon tax according to similar criteria, though the computation rules are different. Finally, John's company would also like to track John's moves to organize his rounds better. GPS raw data is obviously highly private. Fortunately, John's PDS externalizes only the relevant aggregated values to each application. In other words, each application is granted access to a particular view of the data registered in John's database.

3.3 BestLoan.com & BudgetOptim scenarios

Alice needs a loan to buy an apartment. She would like to find the best rates for her loan and, thus, relies on the service of *BestLoan.com* (BL for short), a mortgage broker. To assess Alice's financial situation, BL needs to get access to sensitive information from Alice's PDS such as salary, bank statements and tax information. Alice's data can be securely shared with Donald, a BL employee, as follows: (1) Alice opts in for the BL application and downloads the security policy associated to it in her PDS, (2) Donald authenticates to Alice's PDS with his credentials embedded in his own PDS and requests the required data, (3) Alice agrees to share this data with Donald for a specified duration (e.g., two weeks), (4) finally Donald downloads the data in his PDS, all this by exchanging messages and data through the e-Store Supporting Servers. Donald cannot perform actions on Alice's data exceeding their privileges or the retention period fixed by Alice because his PDS will preclude these actions. If Alice distrusts Donald, she can audit his activity and can at any moment opt out of the BL application (with the effect of deleting Alice's data in Donald's PDS), all this again by exchanging messages among PDSs through the e-Store Supporting Servers.

Alice now wants to optimize her budget and thus opts in for the BudgetOptim application (BO for short). BO runs locally on Alice's PDS with a GUI running on the terminal. BO accesses details of Alice's invoices, telecom bills, etc. in order to suggest more advantageous services according to her consuming profile. With BO application, Alice does not share data with anybody. This last scenario is typical of many private applications that can process personal data (e.g., diet advices, tax minimization, pension simulation, vaccine reminders, etc.).

3.4 Positioning

Compared to an approach where all personal data is gathered on a traditional server, the benefit provided by PDS in the scenarios presented above is fourfold. First, the PDS holder is his own Database Service Provider. Hence, abusive uses by the Database Service Provider are precluded. Second, the PDS provides the holder with tangible elements of trust which cannot be provided by any traditional server (see factors 1 to 4 in Section 2). Third, privacy principles (e.g., limited retention, audit) can be enforced for the data externalized by the holder provided the recipient of this data is another PDS. Fourth, the holder's data remains available in disconnected mode.

However, alternatives to the traditional server exist. The Hippocratic database approach [3] has been precisely designed to protect personal data thanks to principles like purpose specification, consent, limited collection, limited retention, audit, safety, etc. Part of our architectural ideas has been inspired by this work. But the Hippocratic database approach provides tangible guarantees only if the server can be fully trusted. In this respect, the PDS approach can be seen as a fully distributed implementation of a Hippocratic database where the founding Hippocratic principles can be definitely enforced.

The Database as a Service approach (DAS) [18] is another option. Here data is stored encrypted on the server and is decrypted at the client side, making server attacks harmless. This time, the DAS approach makes sense only if all clients can be trusted. Again, part of our ideas has been inspired by the DAS work and the PDS provides a way to make the clients trusted.

Statistical databases [1] and data anonymization [14] are both motivated by the desire to compute statistics or to mine data without compromising sensitive information about individuals. Both require trusting the server, either to perform query restriction or data perturbation in the former case, or to produce the anonymized data set in the latter case. Though orthogonal to the PDS approach, these concerns still exist in the PDS context and must be addressed adequately.

4. PDS GLOBAL ARCHITECTURE

As mentioned in the introduction, PDS is not a simple secure repository of personal documents but rather provides a well organized, structured, consistent and queryable representation of these documents for serving applications requests. The difficulty to achieve this objective comes notably from the variety of data sources and applications targeted by PDS. This section presents an initial design of the PDS architecture considering a set of simplifying assumptions.

4.1 Personal database

In this design, we assume that the DBMS engine embedded in the PDS is relational but the choice of the database model (relational, XML, hybrid) has little impact in the global architecture. Hence, the personal database is assumed to be composed of a small set of relational database schemas, typically one per application domain. We make no assumption on the granularity of application domains but e-health and e-administration are illustrative examples of domains. Database schemas are defined by *DB Schema Providers*. Depending on the domain, a DB Schema Provider can be a government agency (e.g., Ministry of Health) or a private consortium (e.g., a group of banks and insurances).

Content Providers are external information systems that deliver personal data (e.g., blood test, salary form), encoded in XML. We make the simplifying assumption that each XML document conforms to one XML schema defined by a standardization organization (e.g., HL7) or by a DB Schema Provider (e.g., the Ministry of Health). To allow building a consistent and structured view of a set of related documents, an XML document (e.g., a prescription) is enriched with all referential data required to fill the embedded database accurately (e.g., detailed data related to the doctor who wrote the prescription and to the drug prescribed). Hence, the data contained in different documents related to a given doctor or drug can be easily retrieved by SQL queries and cross documents processing becomes possible (e.g., get the list of current medical treatments or compute average blood pressure during the last month). Then the enriched document is pushed in an encrypted form to the recipient PDS through Supporting Servers (see section 4.4 for a description of Supporting Servers). The recipient PDS downloads the XML document and wraps it into a set of tuples thanks to *mapping rules* provided by DB Schema Providers[3]. Mapping rules are declarative and interpreted by a *generic wrapper*, a certified component of the PDS core (see Section 4.5 for a deeper discussion on certification). The benefit of declarative mapping rules is not only that it simplifies the work of the DB Schema Provider but primarily that the safety of these rules can be controlled.

Figure 3 illustrates the wrapping of a prescription, enriched with doctor and drug referentials sent from a hospital. The document conforms to an XML schema for healthcare, and is wrapped into four tables (two of them being referentials) from the healthcare database schema. As shown in Figure 4, not all documents are wrapped and integrated in the database. Some documents (e.g., an X-ray image) can stay encrypted in the Supporting Servers and simply be referenced by the embedded database.

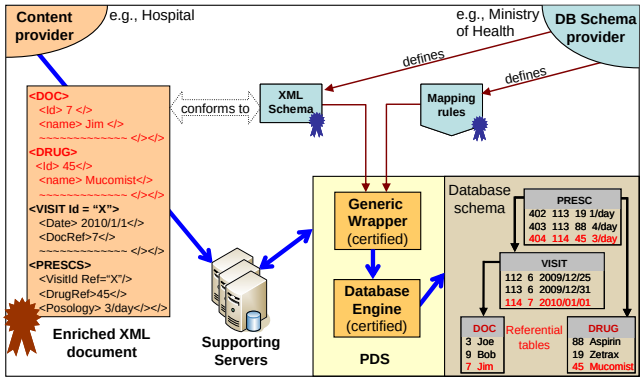


Figure 3: Wrapping a document into the PDS database

Note that problems incurred by the existence of several standards in a given application domain and the problems of data redundancy when database schemas overlap are orthogonal to the PDS approach and are not tackled in this paper.

4.2 Applications

Applications are developed by *Application Providers* (e.g., BestLoan.com). They are defined on top of the published DB schema(s) of interest and can use all database functionalities provided by the embedded DBMS (i.e., DDL and DML statements). Each application defines a set of *collection rules* specifying the subset of documents required to accomplish its purpose (e.g., the five most recent salary forms are required by BestLoan.com). These rules are expressed at the document level to make them meaningful to the PDS holder (helping him to opt in or opt out of this application) and are mapped at the database level to be enforced similarly to access control rules. Applications can run locally (on the holder's PDS with a GUI on a terminal), on another user's PDS (e.g., on the doctor's one) or on an external server sending queries to the holder's PDS (e.g., the Treasury server computing the holder's carbon tax). While most applications are assumed to perform only selection queries, insertion of new documents is not precluded (e.g., a treatment prescribed at home by the doctor). An updating application will play the same role as a content provider and the insertion will follow the same process.

[3] The mapping rules are related to the transcription of documents into a structured database and would be required even with an XML database.

4.3 User Control

The prime way for the PDS holder to control the usage of her data is to opt-in/out of applications and to decide situations where she physically delivers her PDS to another individual (e.g., a doctor). Assuming that the PDS holder's consent has been given, the actions that any individual can perform are regulated by a predefined access control policy. This policy can either be defined by the DB schema provider (e.g., the Ministry of Health fixes a RBAC policy stating the privileges of each category of professionals according to current legislation) or be defined by the Application Provider and be ratified by a consumer protection association or the legislator.

Predefined access control policies are usually far too complex to be understandable by the PDS holder (e.g., the RBAC matrix regulating the use of the French EHR contains more than 400 entries). It is therefore mandatory to provide the PDS holder with simple tools to protect her sensitive data following her wish. A first way consists in managing the authorizations through a simple GUI, as illustrated in the healthcare scenario. A second way is to give the user the ability to mask documents in the database. The tuples corresponding to a masked document are no longer considered at query execution time, except if the query is issued by the PDS holder herself (through an application). To make this process intuitive, the DB Schema Provider can predefine *masking rules* (e.g., hide documents by doctor, pathology, time period, etc.) exploiting the expressive power of the DBMS language and easily selectable by the user through a GUI.

The PDS holder (called hereafter the donor) can also impose privacy preserving rules whenever data leaves her PDS to enter another PDS. This sharing is required when a donor's data must be made available while her PDS is disconnected (see the healthcare scenario). This sharing must be ruled by the following principles:

- *Minimal exposure:* in a nominal use, only the results of authorized queries are externalized by a PDS and raw data always remains confined in the PDS. When donor's raw data is made available to others, this must be done in such a way that minimal data (limited collection principle) is exchanged during a minimal duration (limited retention principle) and with the minimum number of recipient PDS (need-to-know principle) to accomplish the purpose of this externalization.
- *Secure delete:* if the donor decides to delete a document before the retention period expires, all replicas of the corresponding raw data hosted by the recipient PDSs must be deleted.
- *Audit:* the donor must have the ability to audit the actions performed by all recipient PDSs on replicas.

Minimal exposure can be implemented by a Secure Publish/Subscribe mechanism working as follows. The raw data to be exchanged (published) is the tuples belonging to the database view computed over the data targeted by the purpose of the sharing, by intersecting the collection rules of the application, the predefined access control rules applied to the subscribers and the donor's masking rules. The donor publishes these tuples in an encrypted form on the Supporting Servers. The recipient PDSs subscribe to this data and receive the decryption key once the publisher has accepted the subscription. If the content of the view evolves in the publisher PDS (e.g., because new documents have been inserted), the update is pushed to the subscriber PDSs. We assume that publisher and subscriber PDSs have a compatible database schema (e.g., doctors and patients share a uniform healthcare DB schema).

In the following, we denote by *user's control rules* all rules which can be fixed by the PDS holder herself to protect her privacy, namely masking rules, retention rules and audit rules. User's control rules are enforced by all PDSs, both on the PDS holder's data and on the data downloaded after a subscription.

4.4 Supporting Servers

Supporting Servers Providers provide storage and timestamp services to implement the functions that PDSs cannot provide on their own, namely:

- *Asynchronous communication*: since PDSs are often disconnected, documents, shared data and messages must be exchanged asynchronously between Content Providers and PDSs and between PDSs themselves through a storage area.
- *Durability*: the embedded database must be recovered in case of PDS loss or destruction. The PDS holder's personal data can be recovered from the documents sent by Content Providers through the Supporting Servers (assuming these documents are not destroyed). Data downloaded from other PDSs can be recovered from the data published in the Supporting Servers (assuming their retention limit has not been reached). Other data (user's control rules definition, metadata built by applications, etc.) must be saved explicitly by the embedded DBMS on the Supporting Servers (e.g., by sending a message to itself).
- *Global processing:* a temporary storage area is required to implement processing combining data from multiple PDSs. Statistical queries and data anonymization are examples of such processing.
- *Timestamping*: the SPT hardware platform is not equipped with an internal clock since it takes electrical power from the terminal it is plugged in. Hence, auditing and limited retention cannot be implemented without resorting to a secure time server.

4.5 Security

The security of the architecture lies in (1) the tamper-resistance of the SPT platform, (2) the certification of the embedded code (and ratification of declarative rules), and (3) the encryption of any data externalized in the Supporting Servers.

Regarding encryption, we assume that the security of data embedded in a given PDS is comparable to the security of the same data stored encrypted in the Supporting Servers, if the key remains confined to this PDS. Even if any data stored in the Supporting Servers is encrypted, the identity of the users downloading and uploading this data must be obfuscated. Indeed, spying communications could lead to disclosure of sensitive information (e.g., the volume of data sent by a hospital may reveal a heavy pathology). The Supporting Servers provide the storage required to make the communication asynchronous and the PDS themselves integrate a protocol making these communications anonymous.

The certification does not apply to all parts of the embedded code. Typically, assuming the certification of all embedded applications is unrealistic. Figure 4 shows the elements for which certification is mandatory, namely: (1) the core software (operating system, database engine), (2) the generic XML wrapper, (3) the communication manager, (4) the Publish/Subscribe manager and (5) the privacy manager enforcing the user's control rules. Implementing these software pieces and certifying them is the responsibility of the *PDS Providers* (e.g., a SPT manufacturer like Gemalto). Declarative rules need also to be ratified to prove their conformance to a public specification. This data is: (1) the mapping rules consumed by the wrapper, (2) the predefined access control rules, the predefined masking rules and the collection rules enforced by the DBMS. The documents themselves are assumed to be signed to prove their authenticity.

Trusting the predefined access control policies requires being able to authenticate all users. Depending on the application domains, PKI infrastructures already serve this purpose. For example, in France, all healthcare professionals have a certificate embedded in a smart card containing their identity and role (a strong authentication is mandatory to access any server hosting healthcare data). In the same spirit, several countries are developing infrastructures based on smart cards or on software certificate to allow any citizen to authenticate electronically (e.g., IdéNum in France).

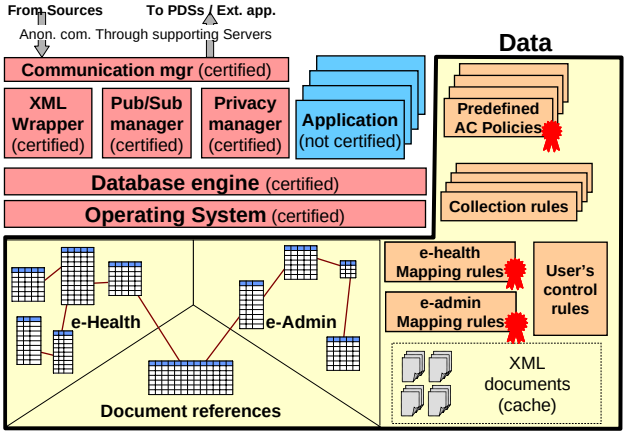


Figure 4: PDS generic software, application, and database

5. EMBEDDED DATA MANAGEMENT

The SPT hardware constraints presented in Section 2 introduce three main technical challenges discussed above.

Computing SQL queries on Gigabytes of data without relying on external resources

Select-Project-Join-Aggregate queries must be executed on Gigabytes of data with Kilobytes of RAM. Join is the most RAM demanding operation. It is usually not supported in tiny RAM devices (e.g., sensors) while it is a central operator in the PDS context. The performance of "Last resort" join algorithms (block nested loop, sort-merge, Grace hash, hybrid hash) quickly deteriorates when the smallest join argument exceeds the RAM size [17]. Jive join and Slam join use join indices [20] but both require that the RAM size is of the order of the square root of the size of the smaller table. In the PDS context, swapping data in the terminal or in the local NAND Flash is precluded due (1) to the dramatic amount of swapping required considering the ratio between the RAM size and the potential sizes of the tables to be joined and (2) to the cost of encryption (only the microcontroller is trusted).

Consequently, the unique solution is to resort to a highly indexed model where all (key) joins are precomputed. In [7], we already proposed a multiway join index called Subtree Key Table and a Climbing Index allowing to speed up selections at the leaves of a join tree. Combined together, these indexes allow selecting tuples in any table, reaching any other table in the join path in a single step. Queries can then be executed in a pure pipeline fashion without consuming RAM or producing intermediate results. This work must be considered as a first step towards the definition of indexing models and query execution techniques dedicated to tiny RAM devices.

Efficient atomic storage and indexing model in NAND Flash

NAND Flash chips exhibit uncommon characteristics: (1) reads and writes are done at a page granularity, but writes are more costly than reads, (2) a page cannot be rewritten without erasing the complete block containing it, which is a costly operation, (3) writes must be done sequentially within a block and (4) a block wears out after about 10^5 repeated write/erase cycles. A main consequence of this is that random writes can be up to order(s) of magnitude more costly than sequential writes [10]. Combining these constraints with the RAM limit makes the storage and indexing problem very challenging.

Regular indexing techniques (e.g., B+Tree) are poorly adapted to NAND Flash because of the high number of random writes they incur [24]. All improvements (e.g., BFTL [24], Lazy-Adaptive Tree [2]) rely on the idea to defer index updates using a log (or Flash-resident cascaded buffers) and batch them to decrease the number of writes. The side effect is a higher RAM consumption (to index the log or to implement write-coalescing of buffers) and a waste of Flash memory space.

A suggested alternative is to try to organize the whole database in a pure sequential way to take advantage of the update pattern of PDS (massive insertions, almost no updates, and few deletes) and of the Flash characteristics. The benefit of sequentiality is in minimizing the need for buffering and caching (thereby saving RAM), in avoiding random writes and in greatly simplifying transaction atomicity because only a set of high watermarks have to be maintained to determine whether NAND Flash blocks contain dirty data or not. Updates and deletes are not reported on the database. Rather, they are kept in a sequential list, the updated pages are marked and their up-to-date image is rebuilt on the fly when the page is loaded in RAM, in a way inspired by [19]. The index problem is more complex since even sequential insertions generate random updates in the index. In [26], we suggested a Flash-aware indexing technique, called PBFilter, which organizes also the index sequentially and speeds-up lookups thanks to partitioned Bloom filters. However, this strategy does not scale for GB of data. To tackle this problem, we are investigating a solution where the database is stratified so that the indexing strategy can change among strata without incurring a dramatic number of rewrites. We feel that designing storage and indexing techniques combining the Flash constraints and the embedded constraints (RAM limitation, optimal Flash usage) deserve a great interest considering the increasing diversity of Flash-based devices.

Enforcing local data confidentiality and integrity

The NAND Flash being not protected by the tamper-resistance of the microcontroller, cryptographic techniques must be used to protect the database footprint against confidentiality and integrity attacks. Indeed, integrity attacks make sense because the PDS holder herself can try to tamper the database (e.g., she could perform a replay attack to be refunded several times for the same drug or try to change an access control rule or the content of an administrative document, e.g., a diploma).

A primary concern in the PDS context is the granularity of the traditional encryption and hashing algorithms (e.g., 128 bits for AES and 512 bits for SHA). As explained above, the PDS query execution engine must rely on a highly indexed model, thereby generating very fine grain random accesses (in the order of the size of a pointer). Solutions to this problem can be: (1) designing encryption and hashing techniques for fine grain accesses [21] compatible with the SPT's resources, (2) designing clustering techniques so that relevant data are contiguous, in the spirit of the PAX models [4] and (3) encrypting the data in such a way that lookups can be done without decrypting the data. The idea here is different from order-preserving encryption or privacy-homomorphism. Roughly speaking, the idea is to exploit the sequentiality of the database to encrypt the data according to their insertion order (hence data having the same clear text get a

different cipher text) but equality tests on the cipher text remain possible if they take this order into account. Version management, required to detect replay attacks, is another complex issue. Maintaining a version number for each page in secure storage (i.e., in the NOR of the microcontroller) is unrealistic considering the small size of the NOR and the fact that it is primarily dedicated to the storage of application code. TEC-Tree [12] overcomes this problem by organizing secret information as a tree. However, it incurs update propagation in the tree, which badly adapts to NAND Flash. Again, our expectation is that the sequential organization of the database can lead to smarter version management techniques. Hence PDS introduces specific interesting challenges in terms of cryptographic techniques applied to database management.

6. DURABILITY, AVAILABILITY AND GLOBAL PROCESSING

Durability and Availability

Honest but Curious Supporting Servers are assumed to correctly store, retrieve and delete data requests on an unbounded storage area in a durable and highly available way. PDSs capitalize on this to implement higher level secure functions.

Anonymous communications between Content Providers and PDSs and between PDSs themselves can be implemented through the Supporting Servers using an anonymizing network like Tor [11], based on the Onion-routing protocol [16]. The anonymizing network provides a virtual circuit *C* from the source to the Supporting Servers. Thus, the latter can send data back to the source without knowing its identity, following the *return circuit* C^{-1} encoded in the initial message (this is called *Reply Onions* [16]). An interesting challenge is to use the secure microcontrollers of SPTs to increase the security of anonymous protocols, having SPTs as entry or exit point for the anonymous route.

Recipient PDSs must be able to retrieve messages or data sent to them. Although communications are anonymous, the difficulty lies in selecting the relevant message/data without disclosing any information that could help the Supporting Servers to infer PDS identity. A protocol tagging messages with anonymous markers is proposed to this end. The delete request is trickier to implement. First, the physical image of the targeted data should be destroyed by the Supporting Servers (e.g., for cleaning purpose) only if the requesting PDS can exhibit an anonymous proof of legitimacy for this request. Second, the deletion must be effective even if an attacker spies all messages sent to the Supporting Servers and records them. Hence, there is no other solution than removing definitely the access to some data (i.e., by removing the way to decrypt it) even if its image has been stolen and cannot be physically destroyed. To tackle this problem, we defined a protocol based on the Diffie-Hellman key agreement. Note that secure deletion is also a prerequisite to enforce masking and limited retention. Assuming Supporting Servers guarantee the durability of all messages/data sent to them (except those legitimately destroyed), the log enabling PDSs to recover after a crash or a loss comes for free. Finally, enforcing audit requires a protocol guaranteeing that audit logs are produced and delivered despite unpredictable disconnections of the subscriber and the publisher PDSs. An initial version of the main protocols is given in Annex C.

Global processing

Executing global processing over a set of autonomous trusted PDSs connecting to Honest but Curious Supporting Servers leads to unusual computations in order (1) to tackle the unpredictable nature of PDS connections and (2) to preserve PDS holders' privacy. We illustrate this through examples.

The Ministry of Health would like to prevent a pandemic. It executes a continuous-like query on each PDS that connects to the Supporting Server in order to select individuals having a given set of symptoms. If more than *p* individuals living in the same region are at risk, they are encouraged to go to a hospital. However, the patients consent to this form of dynamic queries only if their anonymity is guaranteed. The query can then be of the form 'SELECT *pseudonym, city* FROM *any PDS* WHERE *symptom in 'x,y,z''* where pseudonym and city are sent to the querier in the clear through the Supporting Servers. If threshold *p* is reached, the querier sends messages back tagged with the pseudonym of the individual at risk to the Supporting Servers. Thanks to anonymous communication, a PDS holder can get the outcome of the query for herself without revealing her identity. Interesting issues lie in the organization of the continuous querying protocol, in the classification of the queries which can be managed in this manner and in the conditions to preserve anonymity (i.e., anonymity could be breached if successive queries succeed in recomposing the association between quasi-identifiers and sensitive attributes).

Statistical databases [1] aim at answering aggregate queries (e.g., "SELECT *AVG(IQ)* FROM … WHERE *Age=10* AND *Diagnosis='Dyspraxia'*") without compromising sensitive information about individuals. Examples of disclosure control techniques include analyzing the query trail to prevent compromising overlaps between successive queries and/or perturbing the result without affecting the global distribution [25]. An interesting feature of the PDS context is that successive aggregates are computed over a fluctuating population of PDSs (due to the unpredictable nature of PDS connections), making inference among runs harder and influencing the design of disclosure control algorithms accordingly.

Privacy Preserving Data Publishing is another form of global processing aimed at publishing a set of micro-data while protecting the identity of individuals. The traditional process is composed of three phases: data collection, computation of sanitization rules based on the collected data and finally data sanitization. The challenge here is to design a distributed protocol that (1) allows the publisher (through the Supporting Servers) to collect enough data from the targeted PDSs to compute the sanitization rule, and then (2) delegates the sanitization process itself to the PDSs (so that raw data is never exposed) while providing them a way to control the safety of the sanitization rules. We suggested a preliminary solution [5] for a sanitization algorithm preventing record linkages through k-anonymity [22]. Much work remains to be done to prevent from other types of linkages (e.g., attribute linkage prevention through l-diversity [14]).

7. USER CONTROL

Enforcing user's control rules, namely masking, limited retention and audit and combining them with application's collection rules introduce a set of interesting problems described below

Impedance mismatch between documents and databases

While predefined access control rules (e.g., RBAC matrix published by an application or by the DB Schema Provider) and queries issued by applications are expressed at the database level (e.g., in SQL), user's control rules as well as application's collection rules are expressed over documents to be meaningful for the end-user. Conversely, for audit purposes, accesses are recorded at the database level but must be delivered to the end-user at document level in order to interpret them. Consequently, *translation structures* must be integrated in the PDS to store document-to-tuple and tuple-to-document links.

The query engine must integrate these links in the query evaluation in order to compute a result compliant with the application's collection rules, the predefined access control rules and the user's masking rules. The evaluation can be as follows. When a document *D* (e.g., a medical prescription) is inserted in the database, the tuples created at wrapping time reference *D* in the database (tuples related to referentials like doctors and drugs are not concerned). Let S_c be the set of documents targeted by the collection rules of application *A* and S_m be the set of documents targeted by the user's masking rules. When *A* queries the database, the query result includes the document references for each selected tuple *t* and this result is post-filtered to keep only the tuples satisfying ($t \in S_c \wedge t \notin S_m$). Post-filtering can be implemented efficiently in RAM constrained environments using Bloom filters [9].

When a delete request is issued for *D* or when *D* reaches its retention limit, it must be removed from the database. The translation structures are used to identify all tuples related to this document. This includes the tuples referencing *D* either directly (e.g., prescription elements) or transitively (i.e., the referential data like the doctor who does the prescription and the drug prescribed). The presence of referential data in a personal database is sensitive and the related tuples must be removed as well. The difficulty lies in the fact that referential data may be shared by other documents. A garbage collector algorithm[4] must be designed to tackle this problem. The deletion of the targeted tuples can be logical (following the tuple marking process sketched in Section 5) or physical, the latter case being more costly due to the Flash constraints.

Propagating user's control rules to other PDSs

If data has been uploaded on the Supporting Servers by a publisher PDS and downloaded by a subscriber PDS, the user's control rules defined by the publisher must be propagated to the subscriber. Being able to implement the mechanisms presented above on the subscriber PDS requires sending the user's control rules and the translation structures along with the data and forwarding to the subscriber any masking and delete operation performed on the fly by the publisher. Hence, the effect of user's control rules will be the same independently of the location of the data and of the number of replica.

8. CONCLUDING REMARKS

Our belief and hope is that the emergence of new hardware devices combining portability, secure processing and Gigabytes-sized storage will revolution the way people think about management and protection of personal data. The vision proposed in this paper of a secure and portable Personal Data Server is a first contribution in this direction. We have presented an initial design for this vision and have identified a set of important technical challenges related to it. Moreover, Annex A presents an experiment in the healthcare field which prefigures the PDS approach and gives some confidence about the feasibility of converting the PDS vision into reality.

Simplifying assumptions have been made, other solutions could have been envisioned to tackle the identified challenges (e.g., designing an XML embedded DBMS) and new challenges could also have been identified by enlarging the PDS vision. We have considered a highly structured vision of the personal database to support rich applications and we have made strong security assumptions by considering that the PDS is the main element of trust in the architecture. These two options can be debated and reconsidered, opening the way for other exciting research work.

[4] Storing reference counters is badly adapted to the Flash update constraints. An option can be to recompute counters dynamically.

9. REFERENCES

[1] Adam, N. R. and Worthmann, J. C. Security-control methods for statistical databases: a comparative study. ACM Comput. Surv., 1989.

[2] Agrawal, D., Ganesan, D., Sitaraman R., Diao Y. and Singh S. Lazy-Adaptive Tree: An Optimized Index Structure for Flash Devices. VLDB, 2009.

[3] Agrawal, R., Kiernan, J., Srikant, R. and Xu, Y. Hippocratic Databases. VLDB, 2002.

[4] Ailamaki, A., DeWitt, D.J. and Hill, M. D. Data page layouts for relational databases on deep memory hierarchies. The VLDB Journal, 2002.

[5] Allard, T., Nguyen, B. and Pucheral, P. Safe Anonymization of Data Hosted in Smart Tokens, PRiSM Technical Report n° 526, 2010.

[6] Allard, T., Anciaux, N., Bouganim, L., Pucheral, P., Thion, R. Trustworthiness of Pervasive Healthcare Folders, Pervasive and Smart Technologies for Healthcare, Information Science Reference, 2009.

[7] Anciaux, N., Benzine, M., Bouganim, L., Pucheral, P. and Shasha, D. GhostDB: Querying Visible and Hidden data without leaks. ACM SIGMOD, 2007.

[8] Anciaux, N., Bouganim, L., Guo, Y., Pucheral, P., Vandewalle J-J. and Yin, S. Pluggable Personal Data Servers. ACM SIGMOD, 2010. To appear (demonstration paper).

[9] Bloom, B. H. Space/time trade-offs in hash coding with allowable errors. Communications of the ACM, 1970.

[10] Bouganim, L., Jónsson, B. Þ. and Bonnet P. uFLIP: Understanding Flash IO Patterns. CIDR, 2009.

[11] Dingledine, R., N. Mathewson, and Syverson P. Tor: The Second-Generation Onion Router. USENIX, 2004.

[12] Elbaz, R., Champagne, D., Lee, R. B., Torres, L., Sassatelli G. and Guillemin P. TEC-Tree: A Low-Cost, Parallelizable Tree for Efficient Defense Against Memory Replay Attacks. CHES, 2007.

[13] Eurosmart. Smart USB token. White paper, Eurosmart, 2008.

[14] Fung, B. C. M., Wang K., Chen R. and Yu P. S. Privacy-preserving data publishing: A survey on recent developments. ACM Computing Surveys, 2010. To appear.

[15] Gagneux, M. Recommandations de la mission de relance du projet de DMP. http://www.sante-jeunesse-sports.gouv.fr/IMG/pdf/Rapport_DMP_mission_Gagneux.pdf (in French).

[16] Goldschlag, D., M. Reed, and Syverson P. Onion Routing for Anonymous and Private Internet Connections. Communications of the ACM, 1999.

[17] Haas, L. M., Carey, M. J., Livny, M. and Shukla, A. Seeking the truth about ad hoc join costs. VLDB Journal, 1997.

[18] Hacıgümüş, H., Iyer, B., and Mehrotra, S. Providing Database as a Service. ICDE, 2002.

[19] Lee, S. and Moon, B. Design of flash-based DBMS: an in-page logging approach. ACM SIGMOD, 2007.

[20] Li, Z. and Ross, K. A. Fast joins using join indices. VLDB Journal, 1999.

[21] Robshaw, M., Billet, O. New Stream Cipher Designs - The eSTREAM Finalists, LNCS 4986, 2008

[22] Sweeney, L. k-anonymity: a model for protecting privacy. Int. J. Uncertain. Fuzziness Knowl.-Based Syst, 2002.

[23] Want, R., Pering, T., Danneels, G., Kumar, M., Sundar, M. and Light, J. The Personal Server: Changing the Way We Think about Ubiquitous Computing. UbiCom, 2002.

[24] Wu, C., Chang, L., and Kuo, T. An Efficient B-Tree Layer for Flash-Memory Storage Systems. RTCSA, 2003.

[25] Xiao, X. and Tao, Y. Output perturbation with query relaxation. VLDB, 2008.

[26] Yin, S., Pucheral, P. and Meng, X. A Sequential Indexing Scheme for flash-based embedded systems. EDBT, 2009.

Annex A: The DMSP experimental project

This annex presents an experimental project of secure and portable medical-social folder (DMSP in French) [6]. Its goal is to improve the coordination of medical and social cares while giving the control back to the patient over how her data is accessed and shared. The DMSP project fits well the PDS vision but is less ambitious and general. It constitutes, however, a first real-life experience bringing some insights on the benefits and feasibility of managing highly sensitive personal data on Secure Portable Tokens. The goal of this annex is precisely to present these insights exemplifying the PDS approach on a real-case study.

The DMSP project is funded by the Yvelines District and is led by INRIA (the French National Research Institute in Computer Sciences). It involves the University of Versailles, SANTEOS (the provider hosting the French National EHR system), Gemalto (the smart card world leader), ALDS and COGITEY (two gerontology networks), the project being targeted to elderly people.

The ageing of population makes the organization of home care a crucial issue and requires sharing medical and social information between different participants (doctors, nurses, social workers, home helpers and family circle) at the patient's bedside. Server-based EHR solutions are inadequate because (1) the access to the folder is conditioned by the existence of a high speed and secure internet connection at any place and any time; and (2) they fail in providing ultimate security guarantees to the patients, a fundamental concern for patients facing complex human situations (diagnosis of terminal illness, addictions, financial difficulties, etc).

The goal of the DMSP project was precisely to address these concerns. The solution adopted is a simplified and mono-application instance of the PDS architecture.

Patients are equipped with SPTs embedding a personal server to manage their medical-social folder. The form factor of patient's SPT is a USB token. The French law imposes that all professionals strongly authenticate to any server containing medical data thanks to a Health Professional Smart Card. This led Gemalto to develop a specific smart badge (see figure 5) acting both as a smart card reader and as a SPT used for synchronization purpose.

A central server achieves the durability and the availability of the patient's folders (without risk of privacy breach, as discussed next), and imports/exports data from/to the gerontology networks information systems. However, few elderly patients have an internet connection at home. Hence the SPTs of professionals are used to carry synchronization data between the central server and the patient's SPTs thereby implementing a "pedestrian network", the latency of which is linked to the frequency of visits at home.

The patients' folder includes social information such as financial resources or scores measuring possible lack of autonomy, as well as medical data like diagnosis, treatments, and evolution of medical metrics (e.g., weight, blood pressure, cholesterol, etc.).

Database schema
The data stored in DMSP has been modeled in a highly structured way to allow expressing powerful queries and access control rules in the application. For instance, diagnosis and prescriptions produced by the professionals are all wrapped into relational tables, e.g., Professional, Visit, Prescription, and Drug tables. The wrapping principle is similar to the one illustrated in Figure 3 though the wrappers used in DMSP are not generic. The contents of Professional and Drug are referential data shared by several documents.

Transaction atomicity and durability
Transaction atomicity is required for inserting documents, e.g., while inserting a diagnosis and associated prescriptions. Also, processing synchronization files issued from the central server requires atomicity. Transaction durability for updates performed at the patient's bedside is ensured only when the synchronization file reaches the central server through the "pedestrian network".

Access control
Predefined access control rules have been set in conformance with the healthcare RBAC matrix edited by the French government. Authorized views are expressed by select-project-join-agg queries. For example, nurses are granted access to the current medical prescriptions (not to the complete history) to be able to administer the treatment. Health professional cannot access raw social data, and vice-versa, but a reduced set of aggregates is allowed. Different levels of authentications are supported, i.e., strong authentication for professional, login/password for family circle, no authentication for occasional visitor with highly restricted access (access to emergency contacts and to a dashboard to notify some events).

User control
The patient can regulate her privacy by (1) selecting professionals that can access her folder (respecting the predefined access control policy); (2) share part of her folder securely within a restricted trusted circle of professionals; and (3) mask sensitive documents (e.g., the one produced by a given doctor, during a given period of time, for a given pathology). These features were required to make the DMSP acceptable for patients; otherwise, the initiative may have been perceived as an additional transgression of their privacy, equivalent to publishing their former "paper-based" folder on the internet. Consequently, part of the patient's data stored on the central server is encrypted and the cryptographic keys remain confined in the Patient's SPT and in the SPTs of the member of her trusted circle (i.e., in the spirit of the PDS publish/subscribe mechanism but less general and dynamic than this protocol).

Queries and application
The SPTs embed a local Web server, Servlets implementing the DMSP application, a JDBC driver, and the DBMS engine. The DMSP application issues SQL queries to feed the GUI and the embedded query engine computes these queries over views (access controls) and takes into account masking rules. The query and part of the application logic is processed inside the SPT microcontroller (DBMS engine and Servlets). The presentation of the results and additional computations (e.g., plotting curves of cholesterol rates) are done by applets using JavaScript.

An experiment in the field has begun since the end of 2009 and involves 25 practitioners and 100 elderly patients. Both the DMSP application and the DBMS internals will be demonstrated at ACM Sigmod 2010 [8], focusing on some elements mentioned in this paper (e.g., PBFilters, SKT and climbing index).

In summary, the DMSP project can be considered as a prefiguration of the PDS vision, with several restrictions in terms of genericity and dynamicity. However, it already demonstrates the interest of this vision in terms of privacy preservation. The experiment in the field will last during the next 18 months and more ambitious extensions are already foreseen. Typically, the French government has adopted, very recently, a law allowing triggering experiments of healthcare folder managed on secure USB tokens[5] for long term illnesses[6]. This could open future

[5] See: http://www.assemblee-nationale.fr/13/dossiers/dossier_medical_cle_USB.asp

opportunities, relying on the DMSP experience and building on the PDS vision.

Additional information on the DMSP project can be found at http://www-smis.inria.fr/~DMSP/home.php

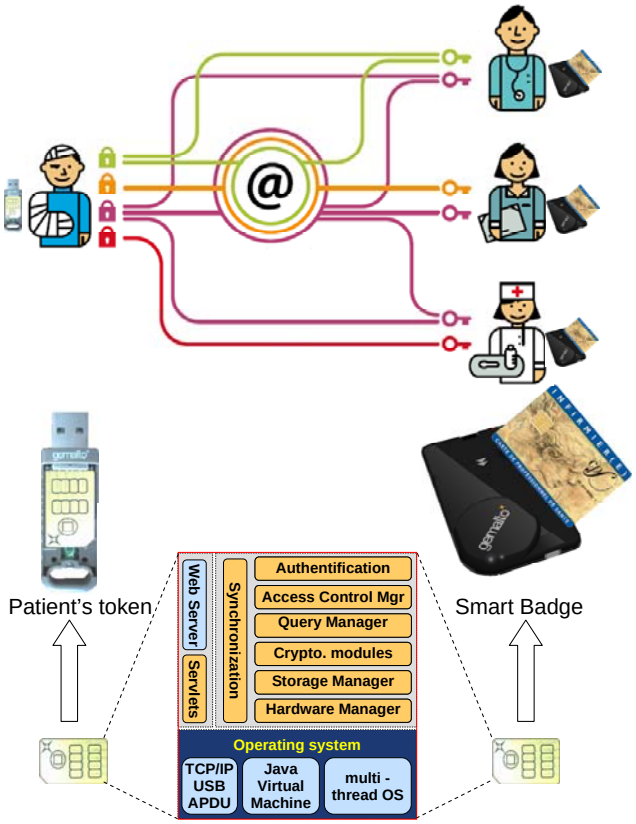


Figure 5. The DMSP project

Annex B: Large scale privacy violations

An example of recent privacy violations caused by negligence:

- The *National Archives and Records Administration* (NARA) is investigating on the loss of a hard drive containing more than 70 millions of veterans' records (social security numbers, dates of birth, names…). The failing hard drive was first sent for repair; however, as the task was too complicated, it was outsourced as-is to another enterprise to be recycled. (DataLossDB, 05 October 2009).

An example of abusive usage of data:

- Attackers who managed to gather hundreds of British medical folders announced that they would sell them £ 4 per unit. They said that several customers – marketing, insurance offices – were interested to sell targeted products to vulnerable people. These medical folders came from a private hospital whose documents were outsourced to an enterprise (which used an Indian subcontractor) in order to be digitized. (The Daily Mail Online, 19 October 2009).

An example of data breaches caused by internal attackers:

- One of the largest reported data breach caused by a malicious insider occurred in 2004 at America Online when 92 million email addresses for 32 million subscribers were sold to spammers. (DataLossDB, Open Security Foundation).

An example of external attacks:

- 30 000 patients of UCSD's Moores Cancer Center have been notified that their personal data - names, dates of birth, medical record number, diagnosis and treatment dates dating back to 2004 – have been leaked after a hacker breached the center's data servers. (Sign on San Diego, 15 July 2009).

Even the most secured servers are not spared:

- 1 600 soldiers have been notified that some personal information, including their names, e-mail messages, phone numbers, home addresses, awards received, ranks, gender, ethnicity and dates of deployment on the field have been breached after an U.S. Army database has been penetrated by unauthorized users. (Federal Computer Week, 12 March 2009).
- A recent computer intrusion that forced the FBI to shut down its computer network and disrupted FBI operations for about 48 hours was traced to an e-mail containing malicious code that originated in China, according to FBI officials. (The Washington Times, 18 June 2009).

Examples of the demands from users for more control:

- 60% of the American people can be considered as privacy pragmatists: they have strong feelings about privacy and are very concerned to protect themselves from the abuse or misuse of their personal information by companies or government agencies. (Alan Westin, Harris Privacy Survey, 2003). Notably, 43% of the people consider that the privacy risk incurred by EHR systems outweighs the excepted benefit. (Harris/Westin survey, 'Privacy and EHR Systems', 2006).
- In the Netherlands, privacy and access concerns are major arguments for the postponement of the national EHR (The International Council on Medical & Care Compunetics, 2009). In particular, the lack of security measures limiting data access for service providers and the loss of control on their own data has been identified as a main reason for citizens to opt-out of the system (within 2 months over 330.000 persons opted-out).

Annex C: Communication protocols

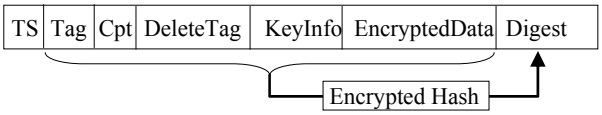
This annex describes the main protocols necessary for anonymously and asynchronously exchanging messages, and supporting deletion. Table 1 introduces a set of notations used in the protocols.

Table 1: List of Symbols used in Protocols

$E_X^{pub}[M]$	Encryption of message M with the public key of entity X.
$E_k[M]$	Encryption of message M with secret key k
$M_1 \| M_2$	Concatenation of messages M_1 and M_2
$H[M]$	Cryptographic hash of message M
$rand_k()$	A pseudorandom number generator using a secret key k, specific to each actor (PDSs or Content Providers)
$ID(X)$	Publicly known identifier of entity X
TS	Timestamp generated by a time server
N	Null value

[6] Considering long term illnesses is of utmost interest since they concern 15% of the population but amount to 65% of the total medical expenses (www.assemblee-nationale.fr/13/rapports /r2347.asp)

Messages sent and stored on Supporting Servers have the following structure:



TS is a timestamp acquired by the supporting server thanks to a secure time server and added to the message to allow filtering out the messages a recipient PDS already received. *Tag* is an anonymous marker allowing a receiver PDS to retrieve its messages on the Supporting Servers. *Cpt* is a counter associated to each sender/receiver pair (or to each marker), incremented by the sender and used by the receiver to check the correctness of the message ordering (not shown in the protocol). *DeleteTag* is a proof of legitimacy for the delete operation, as explained next. *KeyInfo* is a session key used to produce the *EncryptedData* field, itself encrypted with the public key of the receiver. *EncryptedData* is the actual content of the message. Finally, *Digest* is a hash of the previous fields, encrypted with the session key of *KeyInfo* and is used to check the integrity of the message (not shown in the protocols).

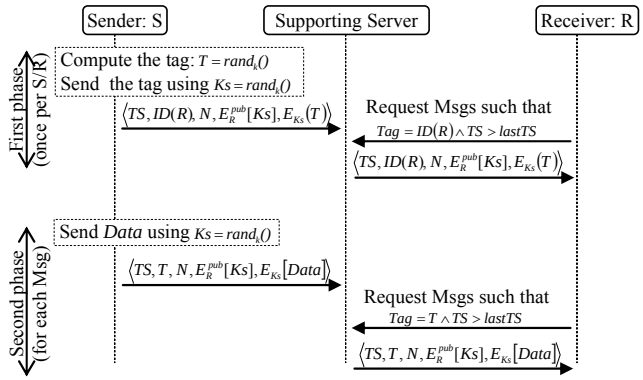


Figure 6. Communication using markers

Message marking and retrieval from Supporting Servers

The protocol to establish anonymous markers works in two phases (see Figure 6). In the first phase, the sender computes a tag *T* (which will be used to tag the next messages) thanks to the pseudorandom number generator. The computed tag *T* is transmitted encrypted with the session key *Ks*, itself encrypted with the public key of the receiver. This first message between a sender and a receiver is itself tagged with the public identifier of the receiver *ID(R)*. Note that, while the receiver identifier is transmitted in clear-text in this first message, it does not disclose sensitive information because (1) the sender is anonymous and (2) for a sender/receiver pair there is only one message of that kind. Hence, an attacker could only count the number of entities who established a communication with a given PDS.

In the second phase, data is exchanged using the defined marker *T*, the timestamp *TS*, and the session key *Ks* encrypted with the public key of the receiver. Note that the reuse of markers with timestamps allows a passive observer to determine that new data items are shared possibly between the same sender and the receiver. Since all communications are anonymous this information cannot be exploited further to link a particular data item to one specific sender or receiver. However, this information could be hidden by changing the marker periodically, transmitting the new marker in the last message using the current marker.

Deletion with proof of legitimacy

A proof of legitimacy is required to guarantee that only the PDS which produces a data can delete it. Audit data is a special case where the PDS which is granted permission to delete some audit data (i.e., the publisher) is actually not the PDS which produces it (i.e., the subscriber). We illustrate below the protocol used when the delete right is delegated to the receiver. The protocol when the sender keeps the delete right can be deduced easily. The idea is based on cryptographic hash functions preimage resistance property. The sender computes a random value called *Delete Proof* or *DP* and applies a cryptographic hash, thus obtaining *DT*, the *Delete Tag*. To transmit the delete right to the receiver, the sender simply adds *DP* to the data before encrypting it. When the receiver receives the message, it extracts *DP* and stores it. At delete time, the receiver sends a delete request, sending *DP* to the Supporting Server. Since given the hash value *DT*, it is computationally infeasible to find *DP*, such that *DT = H(DP)* (pre-image resistance property), the Supporting Server knows that the delete request was sent by an authorized PDS.

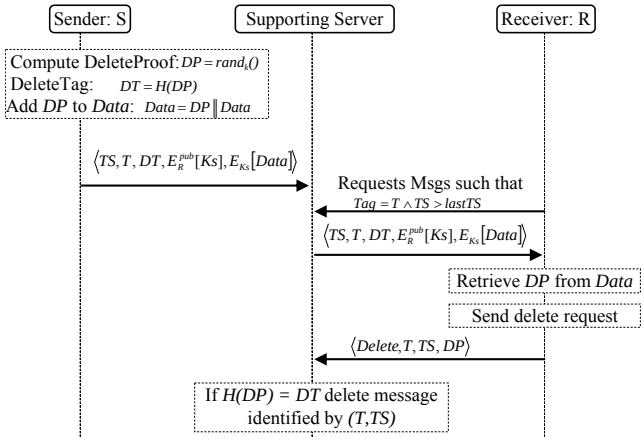


Figure 7. Deletion with proof of legitimacy for the receiver

Secure deletion

All data stored in the Supporting Servers have been carried by messages. Hence deleting a data on the Supporting Servers amounts to deleting the corresponding message. Since the communications may be spied by an attacker and the messages copied, there is no other solution for enforcing the deletion than removing permanently the access to this message. This can be implemented as follows. The sender and the receiver establish a secret key using the Diffie-Hellman key agreement protocol and use it to encrypt the message (thus do not fill the KeyInfo field). When, e.g., the sender decides to delete the message, he destroys his partial secret and sends a message to the receiver requiring deletion of his partial secret. Even if an attacker tampers one of the SPT after the deletion occurs, he cannot recover the message. This idea is simple but the protocol to implement it is more complex due to the fact that each party must be able to recover this message (assuming it has not been yet deleted) in case of a SPT failure (i.e., to ensure the durability property).