# Pluggable Personal Data Servers

Nicolas Anciaux, Luc Bouganim, Yanli Guo, Philippe Pucheral, Jean-Jacques Vandewalle, Shaoyi Yin

# Pluggable Personal Data Servers

Nicolas Anciaux[*], Luc Bouganim[*], Yanli Guo[*], Philippe Pucheral[*,**], Jean-Jacques Vandewalle[***], Shaoyi Yin[*]

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

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

*** Gemalto
La Vigie, La Ciotat, France
<Lastname>@gemalto.com

## ABSTRACT

An increasing amount of personal data is automatically gathered on servers by administrations, hospitals and private companies while several security surveys highlight the failure of database servers to keep confidential data really private. The advent of powerful secure tokens, combining the security of smart card microcontrollers with the storage capacity of NAND Flash chips, introduces a credible alternative to the systematic centralization of personal data. By embedding a full-fledged database server in such device, an individual can now store her personal data in her own secure token, kept under her control, and never disclose in clear her private data to the outside untrusted world. This demonstration shows the benefit of the proposed approach in terms of privacy protection and pervasiveness through a healthcare scenario. This scenario is extracted from a field experiment where medical folders embedded in secure tokens are used to improve the coordination of medical care at home for elderly people. The demonstration also highlights interesting features of the embedded DBMS engine introduced to tackle the secure token's strong hardware constraints.

## Categories and Subject Descriptors

H.2.4 **[Database Management]**: Systems – Query processing
H.2.7 **[Database Management]**: Database Administration – Security, integrity, and protection

## General Terms

Design, Experimentation, Security

## Keywords

Privacy, Secure device, Storage model

## 1. INTRODUCTION

The amount of personal information collected by governments, corporations, commercial Web sites, public and private agencies, is increasing at a high rate. Directives and laws related to the safeguard of personal information slow the flow without stopping it. The suspicion of individuals towards this ineluctable centralization of personal data is merely growing at the same rate. It is fueled by computer security surveys pointing out the vulnerability of database servers against external and internal attacks and also against negligence which led recently to unprecedented sensitive information leakages [4, 5]. Authentication and access control are inoperative in these cases.

Complementary protections can be devised depending on where the trust resides in the system. Hippocratic databases ensure that personal data is used in compliance with the purpose for which the donor gave his consent [3] but require the database server to be trusted. Encrypted databases require either trusting the server or the clients depending on the place decryption occurs [8]. An alternative solution is anonymizing the data [7], assuming the data publisher is trusted, at the price of lesser data accuracy and usability. Finally, databases can be hosted in secure hardware but this solution applied so far only to small mono-user databases.

Today, a new generation of secure hardware devices emerges and drastically transforms the way personal data can be managed. Whatever their form factor (smart card, secure dongle, secure USB stick), *secure tokens* combine the hardware security of smart card microcontrollers, the storage capacity of memory sticks and the performance and universality of the USB communication protocol [6]. Thanks to secure tokens, personal records can be managed under the control of the record owner herself. The use of secure tokens for e-governance (citizen card, driving license, passport, social security, transportation, education, etc) and healthcare (secure personal folders) is actively investigated by many countries. In these initiatives however, a secure token is primarily seen as a raw secure repository, i.e., a set of documents protected by the tamper-resistance of the token and unlocked on demand by the record owner thanks to a PIN code.

This demonstration paper advocates a much broader exploitation of the secure token storage and computing capabilities. It suggests the idea of embedding in secure hardware the complete chain of software usually found on traditional servers, that is a Web server, a set of servlet-based applications and a DBMS engine managing the on-board database and enforcing powerful access control rules. The resulting system, named PlugDB, builds upon a set of research results published earlier [1, 2, 9]. The demonstration scenario itself is based on a real experiment of secure and mobile healthcare folders used to improve care coordination at home for dependent people. Beyond this experiment, we expect that PlugDB will contribute to open new ways of thinking about and organizing the management of personal data.


SIGMOD'10, June 6–10, 2010, Indianapolis, Indiana, USA.

## 2. CONTEXT AND PROBLEM STATEMENT

A Secure Portable Token (SPT for short) combines in the same hardware platform a secure chip and a mass storage NAND Flash memory (several Gigabytes soon). The secure chip is of the smart card type, with a 32 bit RISC CPU clocked at about 50 MHz, memory modules composed of ROM, tens of KB of static RAM, a small quantity of internal stable storage (NOR Flash) and security modules. The mass storage NAND Flash memory is outside the secure chip, connected to it by a bus, and does not benefit from the chip tamper resistance.

Gemalto, the smart card world leader, has developed a JavaCard 3.0 platform for this family of devices which allows the development of embedded Web applications based on Servlet technology. The communication with the external world is supported by the USB 2.0 and the IP protocols. INRIA has developed a database kernel on this platform, providing data storage and indexing, query execution, access control and transaction management. SQL commands are sent to this DBMS kernel through a JDBC bridge. Hence, the SPT with its software suite can be seen as a full-fledged data server accessible through any web browser running on any device equipped with a USB port (laptops, netbooks, PDA and even cell phones). The resulting architecture, called PlugDB, is pictured in Figure 1. Compared to a regular server, a PlugDB server is personal, self-administered, pluggable on demand, does not impose any network connection and provides unprecedented security guarantees.

However, the SPT platform introduces severe hardware constraints. First, security factors imply that the RAM (which has a poor density) must be small – the smaller the silicon die, the most difficult it is to snoop or tamper with processing. Second, the NAND Flash memory exhibits asymmetric costs for reads and writes. Read and writes are done at a page granularity but a page cannot be rewritten without erasing the complete block containing it, which is a costly operation. In addition, a block wears out after about $10^5$ repeated write/erase cycles. Third, a SPT cannot ensure data durability on its own and remote data availability is reduced to the time windows where the SPT is plugged in a terminal linked to the network.

This leads to the following problem statement: (1) how to execute queries with acceptable performance on a Gigabyte-sized database with a tiny RAM, (2) how to design a transactional storage and indexing model tackling the NAND Flash constraints and (3) how to reestablish data durability and availability without compromising security. The next section sketches how these issues have been addressed.

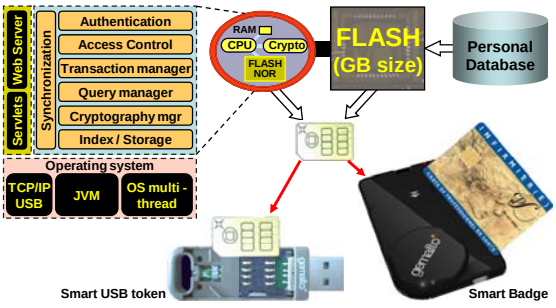


**Figure 1. PlugDB Server**

## 3. TECHNICAL CONTRIBUTIONS

The first challenge to be tackled is to compute regular SQL queries over arbitrarily large tables, with Kilobytes of RAM (the SPT we use holds 64KB of RAM among which 12KB only are devoted to the DBMS) and no resort to swapping (NOR Flash writes are too costly and NAND Flash writes would cause blocks to wear out more quickly and would introduce a security hole unless swapped data is encrypted).

In [2], we shown that last resort join algorithms (like sort or hash joins) as well as known indexing techniques lead to unacceptable performance in our context. We then proposed a new indexing model inspired by data warehouse techniques where the ratio between the database size and the RAM size resemble ours. This model is particularly suited for tree-based database schemas where a root table (e.g., Prescriptions in a healthcare database) references children tables (e.g., Visits and Drugs) in turn referencing sub-children tables (e.g., Doctors and MedicalLabs). Natural joins between these tables are speedup thanks to generalized join indexes called “Subtree Key Tables” or SKT. Each SKT joins all tables in the subtree to the subtree root with the IDs sorted based on the order of IDs in the root table. For example, the entries of the SKT rooted at Prescriptions are of the form (PrescID, VisitID, DoctorID, DrugID, MediclabID) and are sorted on PrescID. This enables a query to directly associate a Prescription with the Doctor who prescribed it and/or the Lab which delivered it, for example.

To speed up selections, we propose an additional index called “climbing index”. A climbing index on a lower table T in the tree-based database schema maps values to lists of identifiers from T as well as lists of identifiers for each table T’ that is an ancestor of T in the tree. Combined together, SKTs and climbing indexes allow selecting tuples in any table, reaching any other table in the path from this table to the root table in a single step and projecting attributes from any other table of the tree. SPJ queries are computed over this indexing model in a pure pipeline fashion following the iterator model.

The indexing model presented above has been proposed in a context different from PlugDB where a static embedded database containing highly sensitive data needs to be combined with public external data sources [2]. The second challenge tackled in PlugDB is then to adapt such a massive indexed scheme to the NAND Flash constraints when updates need to be performed. As personal data servers are likely to gather historical data, audit data, sensed data (e.g., healthcare folders, purchases, visited web sites, bookmarks, geographic locations, etc), tuple insertions are common and must be managed efficiently. Unfortunately, state of the art Flash-based storage and indexing methods were not designed with embedded constraints in mind and poorly adapt to this context. Roughly speaking, they adapt traditional indexing methods by deferring index updates using a log and batching them to decrease the number of rewrite operations in Flash memory [9]. However, all these methods maintain additional data structures in RAM to limit the negative impact of delayed updates on lookup cost. They also perform “out-of-place” updates (usually through a proprietary and opaque Flash Translation Layer) introducing performance unpredictability and reducing Flash memory usage.

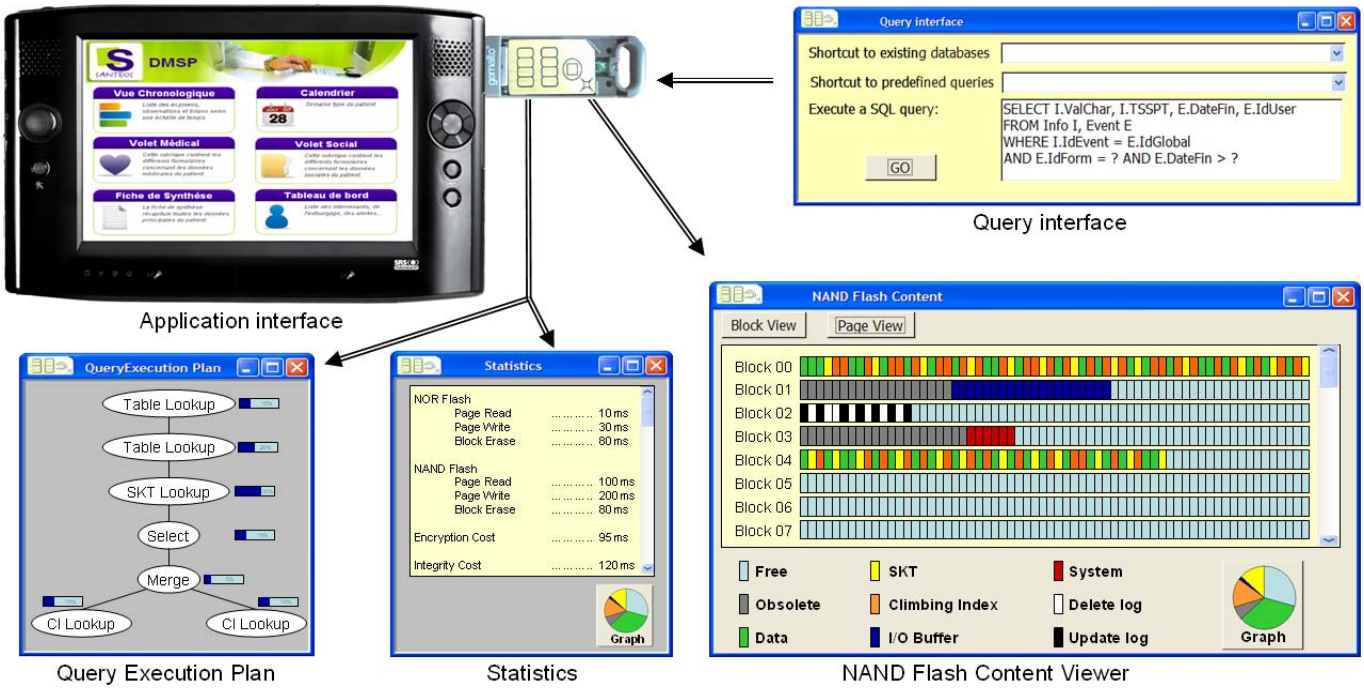


**Figure 2. Demonstration Interfaces**

In [9], we proposed a new alternative for storing and indexing Flash-resident data, called PBFilter, which specifically addresses the embedded context. PBFilter organizes the storage and indexing structure in a purely sequential way to minimize the need for buffering and caching and to avoid the unpredictable side effects incurred by out-of-place updates. But how to look up a given key in a sequential list with acceptable performance? We answered this question using two principles. *Summarization* consists of building an index summary used at lookup time to quickly determine the region of interest in the index list. This introduces an interesting source of tuning between the compression ratio of the summary and its accuracy (i.e., number of false positives). We shown that Bloom filters can be used advantageously as a summarization algorithm, each Bloom filter summarizing all index keys present in a page of Flash. *Partitioning* consists of vertically splitting these Bloom filter summaries in such a way that only a subset of partitions need to be scanned at lookup time (actually one per hash function). This introduces a second trade-off between lookup performance and RAM consumption. The key idea behind Summarization and Partitioning is speeding up lookups without hurting sequential writes in Flash memory. PBFilter gracefully accommodates files up to a few million tuples, a reasonable limit for embedded applications. PBFilter is optimized to support append-oriented files but deletion and updates can be supported with acceptable performance degradation.

The third challenge tackled in PlugDB is to reestablish data durability and availability without compromising security, considering that a SPT is vulnerable (it can be lost, stolen or destroyed) and weakly connected. The solution proposed in [1] is to reintroduce in the architecture a traditional server which will contain only encrypted data. This server can be thought as a highly available extension of the embedded NAND Flash, thereby introducing a third level of stable memory. NOR Flash is the first level of stable memory. It supports high speed reads and benefits from the tamper-resistance of the microcontroller but its storage capacity is low. NAND Flash is the second stable memory level, it provides a much larger capacity, good read performance (much better than a disk though slower than a NOR Flash) but is not hardware protected. Hence, its content must be encrypted to prevent confidentiality attacks and hashed to detect integrity attacks. The encryption/decryption/hashing processes take place in the secure microcontroller. The remote server can store encrypted database archives which can be used to restore the content of a SPT NAND Flash if necessary. It can also store encrypted data the record owner would like to share within a trusted circle of people. In that case, decryption keys are dynamically distributed thanks to a cryptographic protocol among all SPTs participating to this trusted circle. Each one can then access to this data according to its own privileges, in a way similar to accessing data physically stored in their own NAND Flash, at the price of an extra communication with the remote server. This introduces a distinction between three classes of data that the record owner is free to manage according to her perception of the privacy risks: *secret data* remains confined in the owner's SPT and are therefore not durable nor available when the token is unplugged; *durable data* is replicated encrypted on a remote server for durability purpose only; *restricted data* is durable data made available to a trusted circle of people by a secure exchange of encryption keys between SPTs. Note however, that SPT owners never have access to those encryption keys!

## 4. DEMONSTRATION SCENARIO

The demonstration platform is composed of a set of SPTs (one for each member of a trusted circle), a netbook used as a terminal to interact with each SPT and an –untrusted – server used for durability and availability purposes of durable and restricted data.

The first part of the demonstration presents the rationale of the approach and shows the benefits of PlugDB in terms of privacy protection and pervasiveness. This part is based on a real case scenario. Indeed, PlugDB will be experimented early 2010 for eighteen months in the context of a medical-social network providing medical care and social services at home for elderly people (see http://www-smis.inria.fr/~DMSP). The demonstration will show how PlugDB is used in this context to make personal data available in disconnected mode (during a practitioner visit at home), to guarantee data durability (in case of a SPT loss) and to securely share data among the patient, her family doctor and a trusted nurse.

The second phase of the demo focuses on PlugDB's core technology. The objective is to explain the performance observed during the first part of the demonstration by analyzing the queries generated by the application: global queries to display synthesis of a folder, unique select/update to retrieve/insert a particular event in the folder, bulk inserts to synchronize the SPT with a remote server. For each query, the resulting execution plan will be plotted and statistics delivered: CPU and I/O cost per component (access control, query processing, transaction, crypto-protection), highest peak of RAM consumption, NAND Flash usage (number of valid and obsolete pages wrt. total size of the database).

To the best of our knowledge, no system similar to PlugDB exists today and, besides the benefits expected in terms of privacy protection, this demo will give the opportunity to demonstrate exotic core database techniques.

## 5. ACKNOWLEDGMENTS

This research is partially supported by the French National Agency for Research (ANR) under RNTL grant PlugDB and by the French Yvelines District under grant DMSP.

## 6. REFERENCES

[1] T. Allard, N. Anciaux, L. Bouganim, P. Pucheral, R. Thion. 2010. Trustworthiness of Pervasive Healthcare Folders, chapter of the book "Pervasive and Smart Technologies for Healthcare". *IGI Global.*

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

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

[4] Computer Security Institute. 2009. CSI/FBI Computer Crime and Security Survey. http://www.gocsi.com.

[5] DataLoss DB. 2009. http://datalossdb.org.

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

[7] B. C. M. Fung, K. Wang, R. Chen, and P. S. Yu. 2010. Privacy-preserving data publishing: A survey on recent developments. *ACM Computing Surveys 42(4).*

[8] R. Sion. 2007. Secure Data Outsourcing. *VLDB.*

[9] S. Yin, P. Pucheral, X. Meng. 2009. A Sequential Indexing Scheme for Flash-Based Embedded Systems. *EDBT.*