DiSC: Benchmarking Secure Chip DBMS

Nicolas Anciaux, Luc Bouganim, Philippe Pucheral, Patrick Valduriez

DiSC: Benchmarking Secure Chip DBMS

Nicolas Anciaux, Luc Bouganim, Philippe Pucheral, and Patrick Valduriez

Abstract—Secure chips, e.g., present in smart cards, USB dongles, i-buttons, are now ubiquitous in applications with strong security requirements. Moreover, they require embedded data management techniques. However, secure chips have severe hardware constraints, which make traditional database techniques irrelevant. The main problem faced by secure chip DBMS designers is to be able to assess various design choices and trade-offs for different applications. Our solution is to use a benchmark for secure chip DBMS in order to 1) compare different database techniques, 2) predict the limits of on-chip applications, and 3) provide codesign hints. In this paper, we propose Data management in Secure Chip (DiSC), a benchmark that reaches these three objectives. This work benefits from our long experience in developing and tuning data management techniques for the smart card. To validate DiSC, we compare the behavior of candidate data management techniques using a cycle-accurate smart-card simulator. Furthermore, we show the applicability of DiSC to future designs involving new hardware platforms and new database techniques.

Index Terms—Secure chip DBMS, embedded DBMS, benchmark, smart card, access control, access methods, query processing.

1 Introduction

Secure chips, i.e., chips with a high level of tamper resistance, are now ubiquitous in applications with strong security requirements. Secure chips are integrated in smart cards, i-buttons, and other forms of radio frequency or pluggable smart tokens like USB dongles. With more than one billion units sold every year, smart card is the most popular secure chip form factor. It is used worldwide in secured applications such as banking, pay-TV, GSM subscriber identification, loyalty, healthcare, and transportation. Recently, new secure chips form factors have appeared for different applications such as Digital Right Management (DRM) or antipiracy protection for PC (every PC should integrate a Trusted Platform Module (TPM) soon [33]). Although secure chips have very limited computing resources, they are getting more and more powerful. Today, they can support several applications, e.g., using downloaded Java applets. To allow multiple applications to share data efficiently, embedded data management techniques are necessary. Embedding query processing, access control, and transaction management makes the application code smaller and safer. However, secure chips have a unique hardware resource balance (e.g., powerful CPU versus tiny RAM and very fast read versus very slow write in stable storage), making current database techniques, even those designed for lightweight DBMS [15], [22], [29], [32] irrelevant.

In the smart-card context, we addressed the problem of scaling down database techniques and proposed the design of a DBMS kernel called PicoDBMS [24]. We developed a PicoDBMS prototype on a smart-card platform provided by Gemalto, a leading smart-card manufacturer [1]. This prototype has been recently componentized to tackle various application scenarios, and these components have been adapted to the future generation of Gemalto's smart-card platform. In retrospect, building embedded DBMS prototypes stressed the high complexity of developing and validating specific database techniques that encompasses hardware, operating system, and application design issues. With the rapid evolution of hardware and the diversification of embedded applications, it becomes ever harder to select with reasonable confidence the database techniques that best fit a given hardware and software configuration.

Secure chips do not escape Moore's law in terms of CPU speed, memory size, and communication throughput. While the pressure on some resources decreases, the hardware characteristics of secure chips still impose to deeply revisit traditional database techniques. For example, electronic stable memory technologies (e.g., EEPROM or FLASH) exhibit different trade-offs in terms of density, granularity, and performance that strongly impact data management. At the same time, applications on secure chips are experiencing a strong mutation. Secure portable folders are getting growing interest from major industrial players (e.g., MasterCard's Open Data Store [18]), and the introduction of secure chips in common computing infrastructures (e.g., PC or PDA [33]) is paving the way for new large-scale applications. Ambient intelligence is also flooding many aspects of our everyday life, with smart objects gathering information about our habits and preferences. In all of these situations, data hosted by these secure chips is personal and must be carefully protected. Different applications introduce different requirements in terms of access control, and this control has to be performed on chip to make it tamper resistant. Thus, as the access control policy to be enforced on on-chip data gets more sophisticated, the embedded database engine gets more complex, and hardware resource

- *N. Anciaux and L. Bouganim are with the SMIS Project, INRIA Rocquencourt, F-78153 Le Chesnay Cedex, France. E-mail: {nicolas.anciaux, luc.bouganim}@inria.fr.*
- *P. Pucheral is with INRIA and the Laboratoire PRiSM, SMIS Project, INRIA Rocquencourt, F-78153 Le Chesnay Cedex, France. E-mail: philippe.pucheral@inria.fr.*
- *P. Valduriez is with INRIA and LINA, Université de Nantes, 2, rue de la Houssinière, BP 92208, F-44322 Nantes Cedex 03, France. E-mail: patrick.valduriez@inria.fr.*

Manuscript received 18 Dec. 2006; revised 14 Sept. 2007; accepted 11 Mar. 2008; published online 11 Apr. 2008.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-0555-1206.
Digital Object Identifier no. 10.1109/TKDE.2008.67.

1041-4347/08/$25.00 © 2008 IEEE Published by the IEEE Computer Society

consumption increases. For example, select, join, and aggregate operators may be involved in the on-chip computation of an authorized view of the data. Thus, the main problem faced by DBMS kernel designers is to be able to assess various design choices and trade-offs for different applications.

The solution that we propose in this paper is to use a benchmark dedicated to secure chip data management techniques. The objectives of such a benchmark are listed as follows:

1. To compare different database techniques (storage, indexing, access control, and query processing) over different dimensions (e.g., functionality, performance, resource consumption, and code simplicity) in order to select the ones that best match the requirements of a hardware platform/application tandem.
2. To predict the limits (e.g., in terms of performance and database size) of an on-chip application running over a hardware platform/DBMS kernel tandem.
3. To provide codesign hints to help calibrating the resources of a future hardware platform to meet the requirements of on-chip data intensive applications.

Designing such a benchmark is difficult. Existing benchmarks [3], [6], [7], [27], [34] cover different DBMS technologies (relational, object oriented, and XML) and well-established application domains (OLTP, OLAP, ERP, etc.). Moreover, they all focus on performance by considering general-purpose hardware (although the price of the platform is sometimes considered). When dealing with constrained hardware, performance is not the unique concern. Important dimensions like the amount of RAM required to process a query, the database footprint and algorithms' complexity must be captured. Furthermore, it is hard, if not impossible, to define a generic application for secure chip DBMS since new applications keep being invented.

In this paper, we propose DiSC, a benchmark for Data management techniques in Secure Chip. DiSC benefits from our long experience in designing, developing, and tuning DBMS prototypes on different smart-card platforms. DiSC combines five metrics—resource consumption, code simplicity, insertion performance, extraction (of an authorized view) performance, and query performance—in a way that makes reachable the three objectives mentioned above. The benchmark covers well-defined and representative classes of authorizations and captures the consumption of critical hardware resources to enforce them. To validate DiSC, we compare the behavior of candidate data management techniques using a cycle-accurate smart-card simulator.

The paper is organized as follows: Section 2 makes the case for a secure chip DBMS benchmark based on the evolution of hardware and applications. Section 3 defines DiSC with its metrics, data set, and queries. Section 4 introduces a case study of DiSC in the smart-card context. Section 5 reports on the results of benchmarking candidate techniques from this case study using DiSC. Section 6 concludes. An appendix, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2008.67, details the query set and data generator.

2 Case for a Secure Chip DBMS Benchmark

We make the case for a secure chip DBMS benchmark by analyzing the evolution of hardware and applications. Then, we define the requirements for such benchmark.

2.1 Evolution of Secure Chip Technologies

The term secure chip refers to a monolithic chip providing strong antitampering features, whatever its actual form factor (i.e., physical sizes and shapes) ranging from the well-known smart card to chips embedded in smart phones, USB keys, and other forms of pluggable smart tokens. Note that powerful server-based secure coprocessors like the IBM 4758 [10] fall outside this definition. Secure chips share strong hardware commonalities and differ mainly in their interface to the host they connect to [37].

Today's secure chips typically embed on a single chip: a 32-bit RISC processor (clocked at about 50 MHz), memory modules composed of ROM (about 100 Kbytes), static RAM (some Kbytes) and electronic stable storage (hundreds of Kbytes of EEPROM or FLASH), and security modules enforcing physical security. The ROM is used to store the operating system, fixed data, and standard routines. The RAM is used as working memory (heap and stack). Electronic stable storage is used to store persistent information and holds data and downloaded programs. In the following, we analyze the main hardware trends for secure chips highlighting its very unique internal resource balance.

CPU resource. During the last 10 years, embedded processors improved from the first 8-bit generation clocked at 2 MHz to the current 32-bit generation clocked at 50 MHz with an added cryptographic coprocessor to sustain cryptographic computations and enforce security properties. At least three factors justify a continuous growth of CPU power. First, the rapid evolution of secure chip communication throughput (e.g., high delivery contactless cards and USB cards) allows secure chip applications to evolve toward CPU intensive data-flow processing using cryptographic capabilities [23]. Second, many efforts from manufacturers focus on multiapplication and multithreaded secure chip operating systems demanding even more CPU power. Finally, the cost of a chip is mainly driven by the size of the silicon die. In such large-scale markets, this cost consideration strongly favors enhancing the processing power rather than memory, CPU being less surface consuming.

RAM resource. Secure chips hold today a few Kbytes of static RAM, almost entirely consumed by the operating system (and the Java virtual machine in Java chips). The gap between the RAM available to the applications and the CPU and stable storage resources will certainly keep on increasing for three reasons. First, manufacturers tend to reduce the hardware resources to their minimum to save production costs. The relative cell size of static RAM (16 times less compact than ROM and FLASH, 4 times less compact than EEPROM) makes it a critical component [14], which leads to calibrate the RAM to its minimum [2]. Second, minimizing the die size increases the tamper resistance of the chip, thus making physical attacks trickier and more costly. Third, read access to stable storage being fast, traditional cache hierarchy is less mandatory. Since RAM competes with stable memory on the same die, secure chip manufacturers favor the latter against the former to increase chip storage capacity, thus enlarging their application scope.

Stable storage resource. Secure chips rely on a well-established slightly out-of-date hardware technology (0.35 micron) to minimize production costs [28]. However, the market pressure generated by emerging applications leads to a rapid increase of storage capacity. Taking advantage of a 0.18-micron technology allows doubling the storage capacity of EEPROM memories. At the same time, manufacturers are attempting to integrate denser memory on chip, like NOR and NAND FLASH. NOR FLASH is well suited for code due to its eXecute-in-Place property, but its extremely high update cost makes it poorly adapted to data storage. NAND FLASH is a better candidate for data storage though its usage is hard. Reads and writes are made at a page granularity, and any rewrite must be preceded by the erasure of a complete block (commonly 64 pages). In the long term, researchers in memory technologies aim at developing the perfect alternative, i.e., a highly compact nonvolatile memory providing fast read/write operations with fine grain access (e.g., MEMS and PCM). However, integrating these technologies in a secure chip is an even longer perspective due to several difficulties: high security level needs to be proved for any new technology, low manufacturing cost motivates the usage of old amortized technologies, and complexity to integrate all components in the same silicon die.

To conclude, secure chips appear as rather unusual computing environments compared to traditional servers running DBMSs and can be summarized by the following properties: 1) high processing power with respect to the amount of RAM and on-chip data, 2) tiny RAM with respect to the amount of on-chip data, and 3) fast reads but slow and sometimes complex writes/rewrites in stable storage. Identifying such properties for current and future secure chips helps in defining the most accurate and steady metrics for a secure chip DBMS benchmark.

2.2 Evolution of Secure Chip Usage

Smart cards have been used successfully in banking, GSM subscriber identification, or healthcare applications [17]. Now, large-scale governmental projects are pushing for an ever wider acceptance of smart cards (passport, driving license, e-voting, insurance, or transport) in Europe, North America, and other countries [16]. These applications gather a set of persistent data that need to be managed and protected. Secure chips are also at the heart of computing systems to protect PC platforms against piracy [33] or to enforce DRM in rendering devices [31]. Chips are even integrated in a large diversity of usual objects to form an ambient intelligence surrounding.

While we expect secure chips to be almost everywhere in the very short term, the question is whether data management techniques need to be embedded on these devices and for which purpose. To help answering this question, we discuss below two representative application scenarios.

Healthcare folder. The information stored in the future health cards should include the holder's identification, insurance data, emergency data, the holder's doctors, prescriptions, and even links to heavier data (e.g., x-ray exams) stored in hospital servers. Different users may share data in the holder's folder with different privileges. HIPAA specifications consider smart cards as the ideal partner to hold medical data [30]. On-chip data management techniques are mandatory to store, organize, and query this large amount of data and to control the respective privileges of each user.

TABLE 1
Classes of Authorizations for a Secure Chip DBMS

Acronym	Auth°	Authorized view	Required operator(s)
P	SA	Subset of attributes	Project
SP	OA	Predicates on a single relation	Select, Project
SPJ	OA	Predicates on two relations (direct relationship)	Select, Project, Join
SPJ^n	OA	Predicates on several relations (transitive relationship)	Select, Project, Joins
SPG	CA	Single relation, mono-attribute aggregation	Select, Project, Group (aggregate)
SPJG	CA	Several relations, mono-attribute aggregation	Select, Project, Join, Group (aggregate)
SPJG^n	CA	Several relations, multi-attribute aggregation	Select, Project, Join, Group (aggregate)

DRM. Digital piracy is threatening the global multimedia content industry. Basic DRM models fail to solve this problem because they poorly adapt to new attractive usage scenarios and consumers are reluctant to use them for privacy and fairness concerns. Several initiatives [20], [21], [38] demonstrate the need for more expressive DRM languages. Such languages allow business rules to express conditions on historical data, users' profiles, and contextual information. These data must be protected against illegal usage, as well as against tampering from the user herself. Data management techniques embedded in different secure chip form factors (e.g., set-top boxes and smart appliances) are well suited to answer this requirement [5].

Thus, there is a common and growing requirement for on-chip data management techniques, ranging from simple select-project query and access right management up to full-fledged DBMS capabilities depending on the usage. In the sequel, we use the generic term *secure chip DBMS* to capture this diversity of database capabilities.

2.3 Secure Chip DBMS Benchmark Requirements

Secure chip applications share the ultimate goal of enforcing the confidentiality and the integrity of on-chip data. The expected behavior of a secure chip DBMS is to store the data securely (thanks to the chip tamper resistance) and act as a trusted doorkeeper to solely deliver an authorized view of the data to the connected user. This is the main functionality a secure chip DBMS benchmark must capture.

Regarding the access control model, we consider in the benchmark that privileges are granted to users (considering groups or roles would introduce an unnecessary complexity). We also focus on read authorizations since data confidentiality is the primary concern.

Table 1 summarizes the read authorizations a secure chip DBMS should be able to express and enforce. They are expressed over an entity relationship schema. Each authorization type corresponds to a view definition involving one or more relational algebra operators. (For the benchmark, we consider the relational data model since access control over XML databases lacks agreement [9], [11].)

Schema Authorizations (SA) are defined on the database intention (schema) without considering the database extension (occurrences). They are implemented by views involving the project operator (P access right).

Occurrence Authorizations (OA) grant access to occurrences depending on predicates expressed on their properties or on the existence of a relationship (either direct or transitive) with another granted occurrence. The predicates can apply to attributes of one relation (Select-Project access right, named SP), two relations (direct relationship, Select-Project-Join access right, named SPJ), or several relations (transitive relationship access right, named SPJ^n).

Computed data Authorizations (CA) grant access to computed values without granting access to the occurrences taking part in the computation (e.g., averaged values are disclosed but not the raw records). They are implemented by means of views involving aggregates. The aggregation may be monoattribute (i.e., group by on a single attribute) and consider a single relation (Select-Project-Group access right, named SPG), or several relations (Select-Project-Join-Group access right, named SPJG), potentially grouping on several attributes ($SPJG^n$ access right).

To summarize, the requirements for a secure chip DBMS benchmark are the following:

- to cover SA, OA, and CA authorizations with the ability to consider each one separately,
- to capture the consumption of critical hardware resources (namely, stable storage and RAM) incurred by the storage and indexing model and by the operators implementing these authorization classes,
- to capture with the same accuracy the read and update performance of the secure chip DBMS, considering the intrinsic cost and complexity of write/rewrite operations in electronic stable storage, and
- to combine dimensions (functionality, performance, resource consumption, and code complexity) in a way that makes reachable the three objectives mentioned in the introduction, namely, selecting the database techniques that best match a hardware platform/on-chip application tandem, predicting the limit of an on-chip application, and providing codesign hints to calibrate the resources of a future hardware platform with respect to a target application.

Existing database benchmarks do not address these requirements. Early database benchmarks like the Wisconsin benchmark for RDBMS [3] and the OOx benchmarks for OODBMS [6], [7] identify a clear performance bottleneck (join operation in [3] and tree traversal in [6] and [7]) and organize the data set and query set to best capture the behavior of DBMSs in front of this bottleneck [13]. As DBMS technology becomes mature, implementations of different systems converge, thereby making these benchmarks less useful [27]. The Transaction Processing Council [34] overcomes this limitation by publishing a family of benchmarks embedding a database system into specific application scenarios. For example, TPC-C models an OLTP environment, while TPC-H is a decision support benchmark. These benchmarks continuously evolve (some of them becoming obsolete) to remain as a representative of current practices as possible. Going one step further in this direction, SAP provides benchmarks dedicated to a proprietary application [26], the DBMS being evaluated only as a back-end server.

DiSC is in the spirit of the Wisconsin and OOx benchmarks, considering that the database technology of interest is far from being mature. Even supporting join and aggregate operators may be a technical challenge in our context, thus leading to consider simple select-project capable DBMS as possible targets for the benchmark (a rather unusual consideration for standard benchmarks). In addition, all these benchmarks focus on performance. When dealing with embedded software, performance is no longer the unique concern, the most significant dimension among performance, functionality, resource consumption, and code complexity being context dependent.

3 DiSC Definition

In this section, we define the DiSC benchmark with its metrics, data set, and query set. We also show how DiSC meets the three objectives set in the introduction by presenting and analyzing DiSC results as radar charts.

3.1 Metrics

The primary objective of the benchmark is to be able to select, among a collection of candidate database techniques, the ones that best match the requirements of a hardware platform/application tandem. In DiSC, the term database technique refers to the combination of 1) a storage and indexing model to organize the persistent on-chip data, 2) an associated query execution model to compute database views corresponding to one or more of SA, OA, CA authorization classes, and 3) an associated transaction model enforcing ACID executions. We make no assumption on the way data is stored, clustered, or indexed. We introduce five metrics that capture accurately the constraints introduced by the hardware platform and the application: resource consumption, simplicity, extraction performance, query performance, and insertion performance.

Resource consumption. This metric captures the hardware resources consumed by the database technique under test. As discussed in Section 2, the tamper resistance and the production cost of a secure chip are determined by the size of the silicon die, the smaller the better. An embedded database technique influences this parameter by: 1) the quantity of RAM required by its execution, 2) the quantity of XiP-NVRAM (i.e., eXecute-In-Place Non Volatile RAM like ROM or NOR FLASH or any other technology) required to store its code, and 3) the quantity of NVRAM (EEPROM, NAND FLASH, or any other technology) required to store the database footprint it produces (data, associated indexes if any, and logs). Each memory technology exhibits a different density (e.g., RAM cells are four times larger than EEPROM cells and four times larger than ROM cells), and the total silicon surface expressed in square millimeters also depends on the chip engraving technology (e.g., 0.32 or

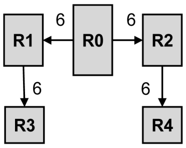


Fig. 1. Generic DB Schema.

0.18 micron). In addition, a database technology may trade RAM for indexes (i.e., indexes may reduce computation complexity) or code footprint for database footprint (i.e., by integrating compression techniques). Thus, we decided to unify all these parameters in a single metric expressed by *Resource (NbTuple/Su)*, where *Su* stands for Surface unit and corresponds to the size occupied by 1 Kbyte of ROM (by convention), while *NbTuple* is the number of entries of the reference database (see Section 3.2 for the details about the data set) related to one *Su*. In other words, *Resource* is the ratio of the total number of database entries divided by the total number of *Su* of required RAM, XiP-NVRAM, and NVRAM altogether.

Simplicity. The objective of this metric is to capture the complexity of the code of the database technique under test. Code complexity is an important concern for embedded secure software due to the cost and time of certification (by exhaustive testing and/or formal proofs) required to guarantee safety and security properties. Code safety is mandatory on mass markets due to the quasi-impossibility to update or patch the software embedded in secure chips. Code security is mandatory by essence since all targeted applications are secure applications. To illustrate this, smart-card software has to comply with the highest certification level (EAL 7) of the Common Criteria, the international standard (ISO/IEC 15408) for computer security [8]. The complexity of certifying software depends on the several parameters that are difficult to capture in a single metric. Thus, while being conscious of this imprecision, we express this metric in terms of the footprint of the database technique implementation: *Simplicity (in Kbytes)* is the DBMS code footprint.

Extraction performance. The performance metric in database benchmarks usually measures the response time of queries over the database. Here, we are mainly concerned with access right management. Thus, it makes sense to measure the performance of the system for building and externalizing complete authorized views rather than for executing queries (the selectivity may be significantly different in both cases). We choose to express this extraction performance of an authorized view in terms of latency and throughput (i.e., like for a storage device) rather than in terms of the usual response time. Indeed, we can view a secure chip DBMS as a smart storage medium: users (either human or applications) connect to it to access the on-chip data (according to their access rights), analogously to any other storage device. In addition, latency and throughput are well adapted to human interaction (e.g., to fill in the window of a mobile terminal hosting the secure chip) and to applications that consume results in pipeline. The latter situation is likely to occur if we consider the discrepancy of computing power between the secure chip and the host it is connected to (e.g., a PC). Thus, we express extraction performance as *Extraction latency (seconds)*, which is the time to get the first tuple of a view, and *Extraction throughput (tuples/second)*, which is the number of tuples of a view produced per second.

TABLE 2
Schema of Relation R0

Attribute name	*Integrity constraint*	*Value type*	*Distinct values*	*Size (bytes)*
A0	Primary key	Numeric	|R0|	4
A1, A2	Foreign key	Numeric	|R0|/6	4
A3		Numeric	|R0|/10	4
A4		Character string	10	20 (avg.)
A5		Character string	100	20 (avg.)
A6		Character string	|R0|/10	20 (avg.)

Query performance. While querying the secure chip DBMS is not the primary concern, it makes sense to measure the performance of querying authorized views. Querying could obviously be delegated to the terminal. However, the secure chip DBMS may take advantage of selection and projection to decrease the cost of externalizing a complete authorized view. Thus, as for extraction performance, the query performance metric is expressed in terms of *Query latency (seconds)* and *Query throughput (tuples/second)*.

Insertion performance. DiSC pays special attention to insert performance for two reasons. First, a secure chip DBMS is likely to manage historical data (medical folder, history of video, and audio assets consumption in DRM applications, history of events in ambient intelligence applications, etc.), a situation where insert is the dominant operation. Second, all forms of NVRAM integrated in secure chips exhibit very poor write/rewrite behavior that must be captured in the benchmark. Insertion performance integrates the time required to create tuples, update indexes if any, check integrity constraints, and perform logging. For the same reasons as before, the insertion performance metric is expressed by the *Insertion throughput (tuples/second)*, number of tuples inserted per second.

3.2 Data Set

As discussed earlier, DiSC does not attempt to be representative of a particular class of applications but strives to provide an adequate framework to compare on-chip database techniques. Thus, we introduce a generic database schema that allows comparing simple to complex queries with variable selectivities. To ease result interpretation, we reduce the generic schema to its simplest form (see Fig. 1). Relations cardinality ratios are similar to the TPC-H benchmark [35], i.e., a large relation (R0) referencing two relations (R1 and R2) that are six times smaller, each referencing in turn a relation (respectively, R3 and R4) that is again six times smaller. R0 is the reference relation and contains as many tuples as the universal relation built by a key join of all relations. The number of database entries mentioned in the Resource metric (see Section 3.1) refers to the cardinality of relation R0.

Table 2 gives the schema of relation R0. Attribute A0 is the primary key, and attributes A1 and A2 are foreign keys referencing relations R1 and R2. Attributes A4 to A6 are variable size strings. A4 (respectively, A5) contains

10 (respectively, 100) distinct values to enable exact selections with a selectivity of 10 percent (respectively, 1 percent), in the same spirit as the data distribution introduced in the Wisconsin benchmark [3]. To assess the compactness of different storage alternatives, A3 (numeric) and A6 (string) take their values in a domain containing $|\mathrm{R0}|/10$ distinct values. Relations R1 to R4 share the same structure as R0, except regarding foreign keys that may be missing. For the sake of simplicity, the attribute names (i.e., numbering) are the same for all relations (e.g., in relation R1, attribute A2 does not exist).

The attribute values generated in this synthetic data set follow a uniform distribution (e.g., each distinct A6 value is shared by $|\mathrm{R0}|/10$ tuples in R0). We discard the idea of generating skewed data distribution for two reasons. First, while real data sets are often skewed, providing realistic skewed schemas is still an open problem and is beyond the scope of this paper. Second, performance measurements are more difficult to interpret when obtained on skewed data. Thus, considering the objective of this benchmark, introducing skewed data is of little interest.

We introduce a Scale Factor (SF) to allow benchmarking a database technique with databases of various sizes. The exact value of $\mathrm{SF} = 1$ may highly depend on the hardware platform under test (from some Kbytes to several Mbytes of stable storage). Since it does not make sense to compare the results of benchmarks conducted on different platforms, the value of $\mathrm{SF} = 1$ is left to the benchmark implementer.

3.3 Query Set

Similar to the data set, the query set has been built with the objective to compare on-chip database techniques rather than to comply with an existing application scenario along with an ad hoc workload. The query set is however representative of the database functionalities required by a secure chip DBMS. Hence, it complies with the access right classification introduced in Table 1. Depending on the authorization classes SA, OA, and CA, different query plans need to be computed from a simple project up to complex multijoins and aggregates. The objective being on comparing database techniques, queries are organized in three groups reflecting the complexity of the query plan rather than the authorization classes.

Group SP contains monorelation queries involving only select and/or project operators. The selection selectivity takes the values 90 percent (to reflect an authorized view extraction), 10 percent and 1 percent (to reflect a query on a view), and 0 percent (to reflect an unsuccessful search). Note that considering very low selectivity (i.e., 90 percent) is rather unusual in database benchmarks but makes sense when extraction of authorized views is considered. Group SPJ contains, key joined, mono- and multijoin queries translating, respectively, direct and transitive relationship authorizations. Selections apply on different relations in the join path from the central relation R0 to peripheral relations and selection selectivity ranges again from 0 percent to 90 percent. Finally, group SPJG contains complex queries involving joins and group-by on one or several attributes with various grouping factors.

The insertion performance metric imposes to consider Insert queries. We consider tuple insertions into R0. Indeed, most insertions occur in R0 since this relation has the highest cardinality. Moreover, insertion in this relation is a worst case in terms of performance since it incurs the highest integrity control cost: two referential constraints need be checked, and the cost of the uniqueness constraint increases with the cardinality of the relation. Thus, we measure the insertion throughput in the last phase of the data set creation (see Appendix B, which can be found on the Computer Society Digital Library at http://doi.ieee.computersociety.org/10.1109/TKDE.2008.67), i.e., R1, R2, R3, and R4 are filled, R0 is created but empty. The throughput is computed as the total time for insertion divided by the cardinality of R0.

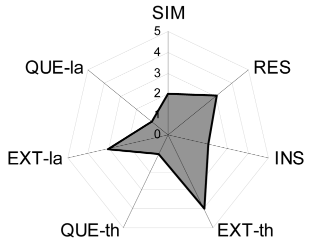


Fig. 2. Example of radar chart.

The complete query set is given in Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieee.computersociety.org/10.1109/TKDE.2008.67. Queries in this set represent either extractions or queries on views depending on the considered selectivity. In the following, we refer to both types of queries, respectively, by the terms extractions and regular queries.

3.4 Benchmark Results as Radar Charts

The relative importance attached to each metric depends on 1) the objective assigned to the benchmark (selecting a database technique, predicting the limits of an application, or calibrating a platform), 2) the intrinsic bottlenecks of the hardware platform under test, and 3) the application requirements. Unlike in traditional DBMS benchmarks where performance is the focus, no metric dominates definitely. Thus, we present the results in the form of radar charts (Fig. 2).

Each radar chart contains the following axes: RES (Resource), SIM (Simplicity), EXT-th (Extraction throughput), EXT-la (Extraction latency), QUE-th (Query throughput), QUE-la (Query latency), and INS (Insertion performance). Each radar chart is plotted for a couple Query Group/SF, where Query Group takes its value among SP, SPJ, and SPJG, and SF represents the size of the database under test. Radars plotted for different values of Query Group should not be compared together. Indeed, they correspond to different alternatives of secure chip DBMS tackling applications with different requirements in terms of access control (in the same way that different benchmarks of the TPC-x family cannot be compared). On each performance axis (INS, EXT-la, EXT-th, QUE-la, and QUE-th), a single point summarizes the measurements performed for all queries of the group. In benchmarks targeting a specific class of applications, ad hoc workloads may be introduced to attach different weights to different queries. DiSC being agnostic to secure-chip applications, the performance measurements are aggregated as follows: Latency is computed as the arithmetical mean of all observed latencies. Throughput is computed as the mean of the observed throughputs weighted by the total duration of the corresponding queries in order to obtain a representative averaged throughput.

Finally, to ease the readability of the radar charts, all axes are uniformly graduated from 0 (worst score) to 5 (best score). The way to translate real numbers—taken from experiments—into axis graduations deserves two remarks. First, establishing a correspondence between score 0 and 5 and absolute minimum and maximum values is actually impractical, simply because most axes are unbounded (e.g., size of the most complex code? maximum insertion, query, or extraction throughput?). Since the objective of DiSC is to compare database techniques, we establish a direct correspondence between score 0 (respectively, score 5) of each axis with the worst (respectively, best) number obtained when measuring the candidate database techniques. Best and worst scores are then relative to a benchmark run. Second, the radar chart should allow to properly visualize the trade-offs between the compared techniques. To this end, different scales may be chosen for different axes. For example, a linear scale could be chosen on a first axis because the measured values are well distributed between the worst and best defined scores, while a logarithmic scale could be chosen on another axis if important gaps exist between the measured values.

Let us illustrate using radar charts how DiSC meets the three objectives set in the introduction. Let us assume first that the benchmark objective is to select the database technique that best matches the requirements of a hardware platform/application tandem. The principle consists in drawing, for all candidate database techniques, the Query Group/SF radar chart that matches the functional application's requirements in terms of access control capability (Query Group) and database size (SF). All radar charts where the resource demand exceeds the hardware platform capacity are discarded. Among the others, the one that offers the best expected compromise between the remaining dimensions is selected.

Let us assume now that the benchmark objective is to predict the limits—in terms of database size under given performance constraints—of an on-chip application running over a hardware platform/DBMS kernel tandem. The principle consists of drawing, for all SFs, the radar charts corresponding to the Query Group of interest for the application and the database technique under test, and then selecting the one that corresponds to the highest SF and achieves the required performance on the dimensions of interest (e.g., insertion, extraction, and/or query performance).

Finally, let us assume that the objective is to calibrate the resources of a future hardware platform to meet the requirements of a target application. The principle consists of drawing for all candidate database techniques, the Query Group/SF radar chart matching the functional application's requirements, then selecting the one that minimizes the hardware resource demand (or the code complexity) while satisfying nonfunctional application's requirements (insertion, extraction, or query performance).

4 Smart-Card Case Study

Through a long lasting cooperation with Gemalto, the world leader in the smart-card market, we have gained a strong expertise in designing and prototyping database components embedded in smart cards [1], [2], [5], [24]. We had to experiment data management techniques of various complexity (SIM phone books, fair DRM engine, and Healthcare folder) on various hardware platforms and simulators. We faced many difficulties to assess our design choices because the metrics of interest tightly depend on the hardware-application tandem. To some extent, these difficulties were the genesis of this benchmark. Thus, assessing the accuracy of the DiSC benchmark by comparing candidate data management techniques developed in this context makes sense. The sequel of this section presents the experimental platform used to conduct smart-card experiments. Then, it introduces the candidate smart-card data management techniques that will be compared in Section 5 using DiSC.

One may ask whether the benchmark should be validated by a comparison of existing embedded DBMS. However, the embedded DBMS that could be compared on the same hardware platform are commercial lightweight DBMS designed for PDA-like platforms. As stated earlier, PDA-like hardware architectures share no commonalities with secure chips, making PDA-like DBMS and secure chip DBMS very different in their design. Conversely, existing on-chip DBMS are not freely available and cannot be compared on the same hardware platform. The reason for this is that on-chip data management techniques are usually embedded in the platform firmware, and their design depends both on the hardware and operating system peculiarities. This is why DiSC has not been designed to compare existing systems but rather to 1) compare data management alternatives for a target platform, 2) predict the limits of on-chip applications, and 3) provide codesign hints.

4.1 Experimental Platforms

Two different platforms have been used to conduct the experiments: a real smart-card prototype and a cycle-accurate hardware simulator of this smart-card prototype (both provided by Gemalto), the latter allowing us to consider data sets exceeding the current smart-card storage capacity.

The real smart-card prototype is equipped with a 32-bit CPU clocked at 50 MHz, 64 Kbytes of EEPROM, 96 Kbytes of ROM, and 4 Kbytes of RAM (with only a hundred of bytes available for the DBMS). An internal timer measures the response time for each incoming APDU (i.e., Application Protocol Data Unit, which is a communication unit between the smart card and the reader as defined in the ISO 7816 standard).

The hardware simulator allows considering data sets up to 1 Mbyte. It is connected to a control PC and plugged into a standard smart-card reader (ISO 7816 standard) connected to a client PC. This hardware simulator is cycle accurate, meaning that it delivers the exact number of CPU cycles between two breakpoints set in the embedded code. Cycle-accurate simulators allow exact performance predictions.

In both environments, the communication cost between the smart card (respectively, the simulator) and the application running on the terminal is not taken into account in the measurements. Indeed, this cost is not a long-term bottleneck (USB smart cards with an 8 Mbits per second throughput are already developed).

The smart-card prototype, as well as the hardware simulator, runs a modified version of the ZePlatform operating system. Based on the experiments we conducted, Gemalto modified the initial version of its operating system to better support data intensive operations (a rather unusual case in traditional smart-card applications). Among others, direct accesses to the EEPROM have been optimized.

4.2 Candidate Data Management Techniques

4.2.1 Common Design Rules

Whatever the target application, the data management techniques under evaluation have been designed with the following objectives in mind:

1. minimization of resource consumption, considering the tiny hardware resources provided by smart cards,
2. code simplicity, considering that software embedded in smart cards is subject to the highest level of certification (common criteria, EAL 7 level),
3. good insertion performance, considering that all target applications we tackled essentially had an append behavior (e.g., secure management of historical data in a medical folder, in a DRM profile, and in a bookmark list), and
4. acceptable, not necessarily optimal, performance for the user in terms of extraction and querying, considering the monouser environment of smart cards.

The complexity of the required onboard data management techniques was tight to the target application and the sophistication of its access control policies. To evaluate dynamically authorized views implementing SA, OA, and CA authorizations on chip, the database components that need to be embedded are a storage manager to organize data and indexes within the chip stable memory, an access right manager to enforce grants and revokes on database views, a query manager to process execution plans, and a transaction manager to enforce the ACID properties. Other database functions (e.g., query parsing and result sorting) do not impact confidentiality and can be executed off card.

To match the smart-card hardware constraints, the design of the on-chip database components follows design rules: Compactness rule (minimize data, index, and code footprint), RAM rule (minimize RAM consumption), Write rule (minimize writes to very slow EEPROM), Read rule (take advantage of very fast reads in EEPROM), Access rule (take advantage of low granularity and direct reads in EEPROM), CPU rule (take advantage of the overdimensioned CPU with respect to the on-chip data), and Security rule (never externalize private data, minimize code complexity).

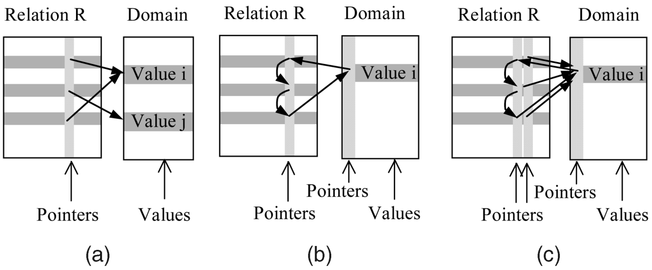


Fig. 3. Storage models for regular attributes. (a) DS. (b) RS. (c) RIS.

4.2.2 Candidate Storage and Indexing Models

Different candidate storage and indexing models can be devised for a smart card. Since the objective is to compare these models using DiSC, we now discuss the trade-offs between them and their respective impact on the processing strategies.

The simplest way to organize data is Flat Storage (FS), where tuples are stored sequentially, and attribute values are embedded in tuples. The main advantage of FS is simplicity and access locality. On the other hand, FS is space consuming (all duplicate attribute values are stored) and relies on sequential scans for all operations. FS may be a good candidate for supporting very simple applications with a low constraint on the database size.

Since locality is no longer an issue in our context (Read and Access rules), pointer-based storage models inspired by main memory DBMS [19], [25] may help combining indexing and compactness. The basic idea is to preclude any duplicate value to occur. Values are grouped in domains (sets of unique values) and attribute values are replaced by pointers within tuples (named tuple-to-value pointers), as shown in Fig. 3a. We call this model Domain Storage (DS). Obviously, attributes with no duplicates (e.g., keys) are not stored using DS but with FS. While tuple update and deletion are more complex than with FS, their implementation could be more efficient in a smart card because the amount of data to be written is smaller (Write rule).

Let us now consider how indexes can be made compact and efficient. A select index is typically made of a collection of values and a collection of value-to-tuple pointers linking each value to all tuples sharing it. The collection of values can be saved since it exactly corresponds to a domain extension. To get the collection of value-to-tuple pointers almost for free, these pointers can be stored in place of the tuple-to-value pointers within the tuples. This yields an index structure that makes a ring from the domain values to the tuples, as shown in Fig. 3b. The ring index can also be used to access the domain values from the tuples and thus serves as data storage model. Thus, we call Ring Storage (RS) the storage of a domain-based attribute indexed by a ring. The index storage cost is reduced to its lowest bound (one pointer per domain value), whatever the cardinality of the indexed relation but slows down access to tuple attributes (project operation) since retrieving the value for the attributes means traversing in average half of the ring (i.e., up to reach the domain value). This extra cost at projection time can be saved by combining RS and DS in a same model called Ring Inverse Storage (RIS; see Fig. 3c).

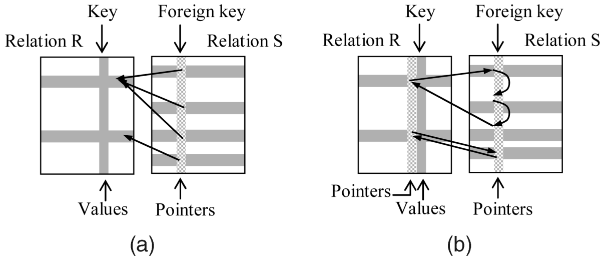


Fig. 4. Storage models for foreign key attributes. (a) DS. (b) RS.

Join indexes [36] can be treated in a similar way. A join predicate of the form $(\mathrm{R.a} = \mathrm{S.b})$ assumes that R.a and S.b vary on the same domain. Storing both R.a and S.b by means of rings leads to define a join index. As most joins are performed on key attributes, R.a being a primary key and S.b being the foreign key referencing R.a, key attributes are stored with FS in our model. Nevertheless, the extension of R.a precisely forms a domain, even if not stored outside of R, and attribute S.b varies on this domain. Thus, DS naturally implements for free a unidirectional join index from S.b to R.a (see Fig. 4a). Traversals from R.a to S.b can be optimized too by RS, a bidirectional join index being obtained by a ring index on S.b (see Fig. 4b). RIS adds direct (i.e., one single pointer) traversals from S.b to R.a to this bidirectional join index.

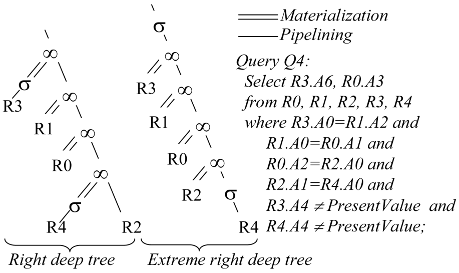
4.2.3 Query Processing

Traditional query processing strives to exploit large main memory for storing temporary data structures (e.g., hash tables) and intermediate results and resort to materialization on disk in case of memory overflow. This hurts both Read and Write rules. To address this issue, we consider query processing techniques that do not use any working RAM area (except for a small collection of cursors) nor incur any write in stable memory.

Let us consider the execution of Select-Project-Join (SPJ) queries. All operators can be combined in an extreme right-deep tree execution plan (see Fig. 5), which leads to pure pipeline execution without materialization. As left operands are always base relations, they are already materialized in stable memory. Pipeline execution can be easily achieved using the well-known Iterator Model [12]. A query execution plan is activated starting at the root of the operator tree. The data flow is demand driven: a child operator passes a tuple onto its parent node in response to a next call from the parent.

The Select operator tests each incoming tuple against the selection predicates. Depending on the storage model, attribute values are directly read in the tuple (FS), reached by dereferencing a pointer (DS/RIS), and/or by following a pointers ring (RS). With RS, the selection predicate (or part of it if multiattribute) can be evaluated on the distinct values of a domain, and the matching tuples are directly retrieved by following the relevant pointers rings.

The project operator is pushed up to the tree since no materialization occurs. Project simply builds a result tuple by copying the value (FS) and/or dereferencing, the cursors (DS/RIS), following the pointers ring (RS) to reach the value present in the input tuple.



```
Query Q4:
  Select R3.A6, R0.A3
  from R0, R1, R2, R3, R4
  where R3.A0=R1.A2 and
      R1.A0=R0.A1 and
      R0.A2=R2.A0 and
      R2.A1=R4.A0 and
      R3.A4 ≠ PresentValue  and
      R4.A4 ≠ PresentValue;
```

Fig. 5. Extreme right deep tree example (Query Q4).

With FS, joins are implemented by nested loops between the left and right inputs, since no other join technique can be applied without ad hoc structures (e.g., hash tables) and/or working area (e.g., sorting). In case of indexes (DS/RS/RIS), the cost of joins depends on the way indexes are traversed. Consider the join between R (n tuples) and S (m tuples), S referencing R through a foreign key. With DS, the join cost is proportional to m starting with S (i.e., right input is S) and to $\mathrm{n}^*\mathrm{m}$ starting with R. With RS, the join cost becomes proportional to $\mathrm{n} + \mathrm{m}$ starting with R and to $\mathrm{m}^2/2\mathrm{n}$ starting with S (retrieving the R tuples associated to each S tuple incurs traversing half of a ring in average). RIS combines the best cases of DS and RS.

Let us finally consider the execution of the aggregate operator (sort is not described since it can be performed on the terminal). At first glance, pipeline execution is not compatible with aggregation, typically performed on materialized intermediate results. The proposed solution exploits two properties: 1) aggregate can be done in pipeline if the incoming tuples are yet grouped by distinct values, and 2) pipeline operators are order-preserving since they consume (and produce) tuples in their arrival order. Thus, enforcing an adequate consumption order at the leaf of the execution tree allows pipelined aggregation. If the grouping attribute is not part of the right leaf relation, the execution tree must be rearranged. Multiattribute aggregation is trickier to implement since it imposes one (respectively, several) Cartesian product at the leaf of the tree to produce tuples ordered by a couple (respectively, tuple) of distinct grouping values.

4.3 On-Chip DBMSs under Evaluation

The objective is to compare the respective merits of each candidate data management technique using DiSC. To this end, we must consider the three Query Groups (SP, SPJ, and SPJG) of DiSC. For each group, we build an adequate on-chip DBMS instance, that is a set of data management techniques that strictly implements the required functionality and should best meet the resource consumption and simplicity expectations (e.g., join and aggregate operators are not required to support SP). For each on-chip DBMS instance, we evaluate the four storage and indexing models (FS, DS, RS, and RIS described in Section 4.2.2), with a common transaction mechanism defined in [24]. This leads to the 12 on-chip DBMS alternatives summarized in Table 3.

Each on-chip DBMS alternative is exploited in order to optimize the access control rights to be enforced. For example, the SP Query Group does not require performing on-chip joins while the SPJ Query Group does. Thus,

TABLE 3
Measured on Chip DBMS Alternatives

Query Group	*Storage index°*	*DBMS version*	*Primary Key (A0)*	*Foreign Key(s) (A1, A2)*	*Numerical attribute (A3)*	*Character string attributes (A4 to A6)*
SP	FS	***SP-FS***	FS	FS	FS	FS
	DS	***SP-DS***	FS	FS	FS	DS (Comp)
	RS	***SP-RS***	FS	FS	RS (Selection)	RS (Comp & Selection)
	RIS	***SP-RIS***	FS	FS	RIS (Selection)	RIS (Comp & Selection)
SPJ	FS	***SPJ-FS***	FS	FS	FS	FS
	DS	***SPJ-DS***	FS	DS (Join)	FS	DS (Comp)
	RS	***SPJ-RS***	FS	RS (Join)	RS (Selection)	RS (Comp & Selection)
	RIS	***SPJ-RIS***	FS	RIS (Join)	RIS (Selection)	RIS (Comp & Selection)
SPJG	FS	***SPJG-FS***	FS	FS	FS	FS
	DS	***SPJG-DS***	FS	DS (Join)	FS	DS (Comp)
	RS	***SPJG-RS***	FS	RS (Join)	RS (Selection)	RS (Comp & Selection)
	RIS	***SPJG-RIS***	FS	RIS (Join)	RIS (Selection)	RIS (Comp & Selection)

foreign key attributes are stored using FS in the SP-RS DBMS alternative and using RS in the SPJ-RS DBMS alternative. The motivations for selecting a particular storage and indexing model for each attribute may be data compactness (Comp), selection performance (Selection), and join performance (Join). Note that SPJ-x and SPJG-x DBMS alternatives share the same storage and indexing model for all attributes.

We believe that the 12 DBMS alternatives under evaluation cover well the spectrum of solutions that could be devised for the targeted context. These alternatives capture the main trade-offs in terms of storage (direct versus compressed), indexation (indexed versus nonindexed structures), and query capability (SP, SPJ, and SPJG). Of course, variations of these alternatives could be considered, but the impact in terms of performance is expected to be low in a main memory like context (the small gap between RIS and RS that will be reported in the next section confirms this allegation).

4.4 Scale Factors

We selected DiSC SFs to cope with a stable storage capacity of 64 Kbytes for the real smart-card prototype (minus the DBMS code footprint located in EEPROM) and 1 Mbyte for the hardware simulator. We define three SFs to model small, medium, and large data sets. The small (respectively, medium) data set is calibrated to fit in the real smart-card prototype (respectively, the hardware simulator) with the less compact storage and indexing model (i.e., FS). The large data set is obtained by excluding FS from the experiments and becomes bound by RIS. The small, medium, and large data sets correspond, respectively, to 360, 9,000, and 14,400 tuples in R0 with a total of 500, 12,500, and 20,000 tuples in the complete data set. In other words, considering the small data set as $\mathrm{SF} = 1$, the medium and large data sets correspond, respectively, to $\mathrm{SF} = 25$ and $\mathrm{SF} = 40$.

5 Comparing DBMS Alternatives with DiSC

This section reports on the results of benchmarking the 12 on-chip DBMS alternatives introduced earlier using DiSC. The performance measurements have been done on the real smart card for the small data set and on the cycle-accurate hardware simulator for the medium and large data sets. Given the large number of DBMS alternatives, we only give synthetic results. In the following, we give the results for each metric of DiSC. Finally, we illustrate the interest of using DiSC radar charts for analyzing the results.

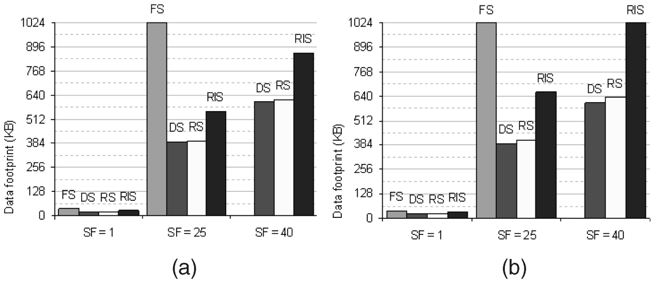


Fig. 6. Database footprint. (a) SP-x DBMS alternatives. (b) SPJ/SPJG-x alternatives.

5.1 Resource Consumption

Resource consumption depends on the database footprint (data, indexes, and logs), the quantity of RAM required by query processing and the DBMS code footprint. Let us first consider the database footprint that is the dominant factor except for very small SFs. Fig. 6 shows the database footprint for each DBMS alternative and SF. Fig. 6a corresponds to SP-x DBMS alternative. Fig. 6b is common to SPJ-x and SPJG-x, which share the same storage and indexing model for all attributes and thus produce the same database footprint.

As expected, FS is less compact since it does not benefit from any form of compression (FS is not measured for $\mathrm{SF} = 40$, see Section 4.4). DS is the most compact model since domains act as a dictionary compression scheme. The extra cost incurred by RS compared to DS is rather small whatever the SF, thus highlighting the high compactness of ring indexes (only one extra pointer per domain value). RIS incurs a much higher storage overhead by adding one pointer per tuple for each indexed attribute (e.g., RS has no impact on the largest relation R0, while RIS adds two pointers per R0 tuple, one for each foreign key occurrence). Thus, RIS should be adopted only if it provides a significant performance gain at extraction and query time.

Let us now study RAM consumption. The pure pipeline query processing strategy presented in Section 4.2.3 reduces the RAM consumption to its lower bound, that is, one cursor per relation and domain involved in the query plan plus one counter per aggregate function to compute. Thus, the impact of RAM consumption on the resource metric is negligible.

Code footprint reduction has been exploited as a marketing advantage by lightweight DBMS vendors. As far as resource consumption is concerned, and except for very small SFs, our experience shows that the challenge is more on reducing the database footprint than the code footprint. This is exemplified by the small difference between the different DBMS alternatives (see Table 4): from 15 Kbytes for the simplest SP-FS up to 42 Kbytes for the most complex SPJG-RIS. Furthermore, the difference between x-FS and x-DS DBMS alternatives is only 4 Kbytes, while the savings on the database footprint are excellent. Finally, even though the DBMS code is located in EEPROM in our prototype, it should be hosted in ROM or in NOR-FLASH in a commercial

TABLE 4
Total Code Footprint

Code footprint (KB)	FS	DS	RS	RIS
SP	15	19	28	31
SPJ	22	26	34	37
SPJG	26	30	39	42

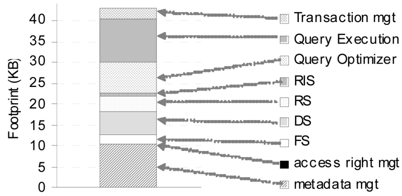


Fig. 7. SPJG-RIS DBMS alternative.

product, both technologies being four times denser than the EEPROM, which hosts the database.

5.2 Code Simplicity

Table 4 gives the total code footprint of each DBMS alternative, while Fig. 7 presents the respective size of each module of the DBMS kernel for the most complete SPJG-RIS alternative. In Fig. 7, three groups of modules can be distinguished, each representing roughly one third of the code: 1) modules related to the data definition (i.e., DDL) including metadata and access right management, 2) modules related to the management of FS, DS, RS, and RIS storage models, and 3) query optimization and execution and transaction management. The rather large size of metadata management is explained as follows: Query execution plans cannot fit in RAM simultaneously, thus precluding to store metadata in database relations and access them by queries. Metadata are therefore stored in ad hoc structures and accessed through ad hoc procedures. Conversely, Table 4 shows that 11 Kbytes of code is enough to implement simple and multijoins, as well as mono and multiattribute aggregations (see difference between SP-x and SPJG-x code footprint). As discussed in Section 4.2.3, the query processing strategy trades a high number of iterations for code simplicity, thanks to the CPU, Read and Access rules.

5.3 Extraction and Query Performance

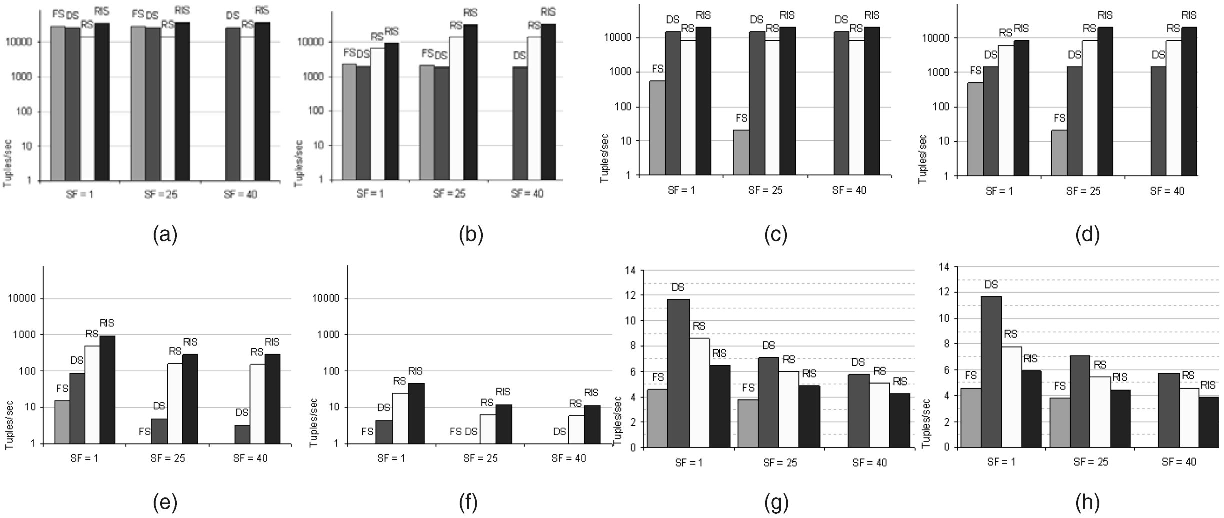
For each Query Group (SP, SPJ, and SPJG) and storage and indexing alternative (FS, DS, RS, and RIS), we show the throughput for extractions and regular queries. Additional curves are plotted for individual queries when required. We omit the latency dimension, which has little interest in our context since pure pipeline execution makes latency roughly equal to 1/throughput. Finally, all diagrams use a logarithmic scale due to the large performance differences among queries. The rather high throughputs are the result of 1) fast and low granularity direct reads in EEPROM (Read and Access rules), 2) overdimensioned CPU with respect to the on-chip data (CPU rule), and 3) the fact that communication cost is not integrated in the measurements (see Section 4.1).

5.3.1 SP Performance

Figs. 8a and 8b show the performance of extractions and regular queries for SP-FS, SP-DS, SP-RS, and SP-RIS (SP-x for short) DBMS alternatives. We make the following observations:

High throughput for all configurations. As expected, the indexed models RIS and RS offer better performance for regular queries compared to FS and DS (Fig. 9b). However, FS and DS outperform RS for extractions (Fig. 9a) because rings slow-down projections (half of the rings must be traversed on the average to get the attribute value).

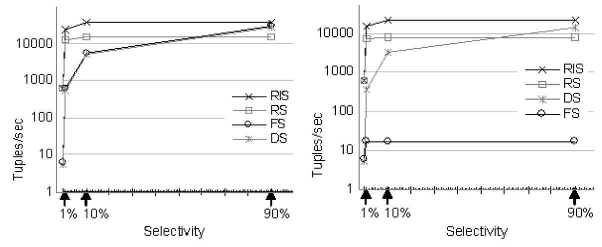
Throughput is independent of the SF. The time required to select and project a given tuple is constant whatever the size of the considered relation. Note, however, that indexed models (RS and RIS) have lower throughput for regular queries with SF = 1 (small data set) since selections require full scan of the domain entries to identify the qualified rings of tuples. For fixed size domains, the relative cost of this step may be important compared to the number of



(a) (b) (c) (d)

(e) (f) (g) (h)

Fig. 8. Performance in extraction, query, and insertion for different DBMS alternatives. (a) SP-x extraction. (b) SP-x regular queries. (c) SPJ-x extraction. (d) SPJ-x regular queries. (e) SPJG-x extraction. (f) SPJG-x regular queries. (g) SP-x insertion. (h) SPJ-x/SPJG-x insertion.



(a) (b)

Fig. 9. Throughput for SP-x query Q2 with $\mathrm{SF} = 25$. (a) SP-x alternatives—Q2. (b) SPJ-x alternatives—Q4.

produced tuples (with $\mathrm{SF} = 1$, attribute A5 holds 100 distinct values, while R0 contains only 360 tuples).

Throughputs suffer from high query selectivity. As shown in Fig. 9, FS and DS throughputs decrease by more than 10 times between extractions (selectivity of 90 percent) and regular queries (selectivity of 1 percent and 10 percent). Indeed, the worse the query selectivity, the less irrelevant tuples scanned and checked before producing a matching tuple. RS and RIS models are much more resistant to selectivity increase, except for extreme values (near 0 percent). For the particular case of 0 percent selectivity, we considered the query response time as the delay to produce the first and unique result (i.e., EOF).

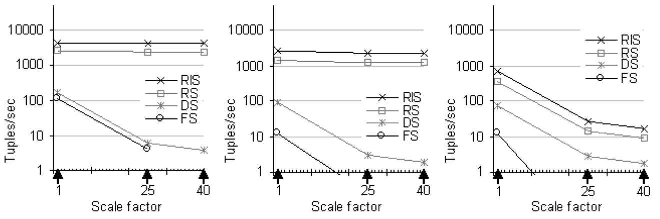
5.3.2 SPJ Performance

The respective performance of SPJ-x DBMS alternatives are shown in Figs. 8c and 8d and lead to the following observations:

Indexed alternatives provide high throughput. FS yields poor performance because join is performed by nested loops. Storing foreign keys with DS naturally implements a unidirectional join index (foreign key to primary key) and yields excellent performance for extractions (even better than RS, which suffers from the projection cost). The performance degrades for regular queries since it imposes a unique join ordering in the query plan, thereby precluding applying selections first. For regular queries, RS is almost 10 times better than DS, and RIS yields the best performance.

Indexed alternatives are independent of the SF. The use of join indexes with DS, RS, and RIS models make their performance independent of the database size. However, as for SP performance, RS and RIS yield lower throughput for regular queries with $\mathrm{SF} = 1$ because of the constant cost of scanning the domain entries to identify the qualified rings of tuples. The performance of FS drastically decreases when the SF increases because of the nested loops.

Throughputs suffer from high query selectivity. SPJ exacerbates the behavior already mentioned with SP, particularly for FS (due to nested loop joins) and also for DS. As mentioned above, using the unidirectional join index provided by DS precludes applying selection on a referenced relation first. This behavior is highlighted in Fig. 9b, which plots the throughput of SPJ query Q4 (joining all relations of the schema) with a selectivity of 0 percent, 1 percent, 10 percent, and 90 percent, respectively, the selection being applied on R1. The throughput of RS and RIS is independent of the selectivity except for very high selectivity. This observation is no longer true for queries with two selections on two different relations since the pipelined query processing strategy precludes applying both selections first. Such specific behaviors are difficult to capture with the benchmark due to the aggregation of all measurements performed within a query group. As discussed in Section 3.4, ad hoc workloads could help tackling this issue assuming a given class of applications is targeted.



(a) (b) (c)

Fig. 10. SPJG alternatives—throughputs for extractions. (a) SPG query. (b) SPJG query. (c) SPJG^n query.

5.3.3 SPJG Performance

This section studies the performance of the SPJG-x DBMS alternatives. Figs. 8e and 8f show the average results for both extractions and regular queries. To allow deeper analysis, Fig. 10 individually plots the result of the three extractions considered in the benchmark, respectively, reflecting SPG, SPJG, and SPJG^n access rights (see Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieee.computersociety.org/10.1109/TKDE.2008.67). The main observations are the following.

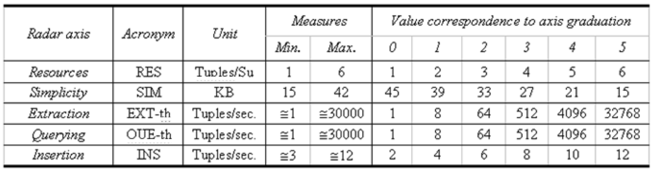
RS and RIS fairly support aggregation. RS and RIS produce in average more than 100 tuples per second for extractions and about 10 tuples per second for regular queries. The important difference between SPJ and SPJG is naturally explained by the high difficulty to handle aggregations without RAM consumption. Thanks to rings, joins within aggregation queries are handled with small degradation (see Fig. 10a versus Fig. 10b). The throughput remains stable when varying the SF for aggregations on a single attribute (Figs. 10a and 10b). For aggregations on two attributes (Fig. 10c), the RS/RIS index can help for only one of them. A Cartesian product is then required, making the performance dependent on the database size.

FS collapses while DS may still fit specific situations. DS and FS exhibit poor throughputs for regular queries and are strongly impacted by the SF. However, DS keeps producing a few tuples per second for extractions (see Fig. 8e), which may be sufficient for some applications.

5.4 Insertion Performance

Figs. 8g and 8h show that the throughput is acceptable (several tuples per second) whatever the DBMS alternative. Differences between Figs. 8g and 8h only appear for RS and RIS and show a small overhead incurred by rings and inverse rings built on foreign key attributes in SPJ-x and SPJG-x.

TABLE 5
Translation Table for Radar Chart Axis Graduations



Radar axis	Acronym	Unit	Measures		Value correspondence to axis graduation					
			Min.	Max.	0	1	2	3	4	5
Resources	RES	Tuples/Su	1	6	1	2	3	4	5	6
Simplicity	SIM	KB	15	42	45	39	33	27	21	15
Extraction	EXT-th	Tuples/sec.	≅1	≅30000	1	8	64	512	4096	32768
Querying	OUE-th	Tuples/sec.	≅1	≅30000	1	8	64	512	4096	32768
Insertion	INS	Tuples/sec.	≅3	≅12	2	4	6	8	10	12

The cost of writes in EEPROM strongly impacts insertion performance. Thus, the smaller the number of writes is, the better the insertion throughput. This explains the differences among all storage and indexing models. The insertion throughput globally decreases when the SF increases with a different proportion for each storage and indexing model. The insertion cost partially depends on the target relation size: while the tuple creation cost (dependent on the storage and indexing model) and the logging cost are independent of the relation size, integrity checking and domain update are dependent on the relation and domain cardinality, respectively.

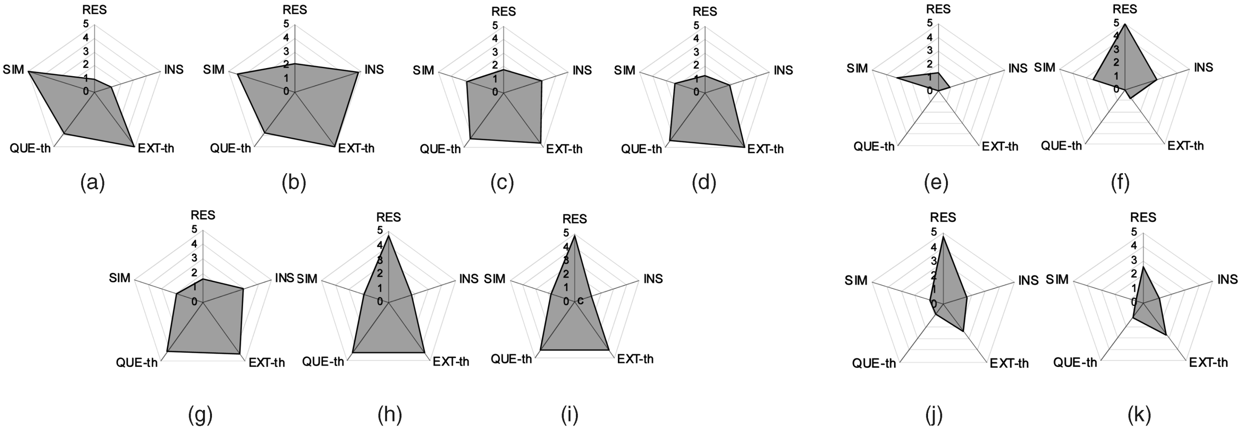
5.5 Using DiSC Radar Charts

This section illustrates the interest of using DiSC radar charts based on the performance results presented above. As explained in Section 3.4, all axes are uniformly graduated from 0 (worst score) to 5 (best score). Thus, the translation of real numbers—taken from the experiments—into axis graduations must be fixed. The scale of each axis must also be fixed, depending on whether a linear or logarithmic scale is more appropriate to quantify the difference between the database techniques (i.e., DBMS alternatives). We choose a linear scale for code Simplicity (SIM axis), Resource (RES axis), and Insertion performance (INS axis). A logarithmic scale is more appropriate for extraction (EXT-th axis) and querying (QUE-th axis) because of the broad intervals among the measured values (logarithm to base 8 is the best suited to score 5 a maximum throughput around 30,000 tuples/second). Table 5 gives the translation between radar chart axis graduations and absolute values.

A large number of radar charts could be plotted, one for each DBMS alternative and SF. We present below three radar chart series identifying interesting trade-offs.

The first series, presented in Figs. 11a, 11b, 11c, and 11d, contains one radar chart per SP-x DBMS alternative for the small data set $(\mathrm{SF} = 1)$. This series is of interest to select the DBMS alternative best suited for very simple applications, where access privileges are reduced to views based on selection and projection, and the data set is reduced to a few hundreds of tuples. Fig. 11a shows that FS is very simple and provides excellent extraction performance and very good query performance but exhibits a poor insertion throughput and high resource consumption. DS is a bit more complex in terms of code but obtains a better resource usage (dictionary-based compression) and reaches the best insertion throughput (integrity checking and domain checking have a negligible cost with $\mathrm{SF} = 1$). RS trades simplicity for even more performance on the query axis but yields a smaller insertion throughput than DS. Finally RIS trades simplicity, insertion throughput, and resources to obtain the best querying and extraction performance. Thus, DS appears as the most attractive alternative and is definitely superior to FS. If extreme query performance is required, RS could be an option but not RIS, which has no advantage in this context. Note that no alternative reaches a good score on the resource axis. The reason is the negative impact of the code footprint on this metric for low SFs.

A second radar chart series is given in Figs. 11e, 11f, and 11g. It analyzes the behavior of SPJ-RS DBMS alternative when increasing the SF. This radar chart series could be exploited to determine the limit of an application requiring access rights on SPJ views and supported by a SPJ-RS DBMS. We are interested in estimating the largest database that can be supported while satisfying given performance constraints. Let us assume that the performance constraints imposed by the application are a minimum average throughput of 5 tuples/second for insertion and 2,000 tuples/second for extraction and querying. In the DBMS alternative of interest, QUE-th and EXT-th axes are independent of the SF and need not be further considered. Thus, the largest database that can be supported turns out



(a) (b) (c) (d) (e) (f)

(g) (h) (i) (j) (k)

Fig. 11. SP-x, SPJ-x and SPJG-x DBMS alternatives.(a) SP-FS $\mathrm{SF} = 1$. (b) SP-DS $\mathrm{SF} = 1$. (c) SP-RS $\mathrm{SF} = 1$. (d) SP-RIS $\mathrm{SF} = 1$. (e) SPJ-RS $\mathrm{SF} = 1$. (f) SPJ-RS $\mathrm{SF} = 25$. (g) SPJ-RS $\mathrm{SF} = 40$. (h) SPJG-FS $\mathrm{SF} = 25$. (i) SPJG-DS $\mathrm{SF} = 25$. (j) SPJG-RS $\mathrm{SF} = 25$. (k) SPJG-RIS $\mathrm{SF} = 25$.

to be $\mathrm{SF} = 25$, due to the imposed score on INS (of course, the more radar charts are drawn, the finer the analysis).

The last radar chart series given in Figs. 11h, 11i, 11j, and 11k concentrates on the SPJG-x DBMS alternatives using the medium data set $(\mathrm{SF} = 25)$. For such application requirements in terms of access control and SF, FS, and DS alternatives offer very poor performance along the extraction and query axes. However, DS might still be of interest—due to its good behavior on the other axes—for specific usage where query and extraction performance in not a concern (e.g., background aggregation, as discussed in Section 5.3.3). In the more general case, RS or RIS are required to achieve reasonable performance. Compared to RS, RIS roughly doubles the query and extraction performance (logarithmic scale) at the expense of 60 percent degradation on resources. Hence, choosing RIS in this setting may make sense if the application is bounded by the performance since the query and extraction performances are globally low.

For the sake of conciseness, we use the same radar chart series to illustrate a codesign scenario. Let us consider that the objective is to determine the amount of hardware resources required to satisfy an on-chip application which has the following requirements: SPJG access control policies, a database size up to $\mathrm{SF} = 25$, an average throughput of 5 tuples/second for insertion, and 50 tuples/second for extraction. FS and DS are too far from the performance objective in terms of extraction. RS consumes much less resources than RIS and is then the DBMS alternative selected. This choice implies that the XiP-NVRAM of the target hardware platform must be calibrated to host at least 39 Kbytes of code. The minimum of NVRAM required to host the database can be easily estimated by $(\mathrm{N/RES} - \mathrm{SIM}) * \mathrm{D1/D2}$, where N is the number of R0 tuples for the SF of interest, RES the resource metric expressed in Tuple/Su $(\mathrm{Su} = \text{surface of 1 Kbytes of ROM})$, SIM the simplicity metric expressed in Kbytes, D1 the density of ROM, and D2 the density of the target NVRAM. Here, $(9{,}000/5 - 39) * 1/4 \approx 450\ \mathrm{Kbytes}$ of EEPROM are required to store the RS representation of the database at $\mathrm{SF} = 25$.

To summarize, radar charts are very useful to visualize the different DiSC metrics on synthetic graphics. This is important to compare different database techniques on different settings and select the one that best matches a given requirement. While a traditional performance analysis would have favored RS and RIS, our analysis using DiSC shows that different application requirements are better fulfilled using other storage and indexing models (e.g., DS).

6 Conclusion

Secure chips have severe hardware constraints that have triggered a deep revisiting of traditional database techniques, with unusual design choices and trade-offs between security, resource consumption, code simplicity, and performance for building a secure chip DBMS. DiSC, result of our long experience in developing and tuning smart-card DBMS prototypes, is a benchmark, which allows 1) to compare different database techniques for secure chip DBMS, 2) to predict the limits of on-chip applications, and 3) to provide codesign hints to help calibrating the resources of a future secure chip to meet the requirements of on-chip data intensive applications.

Our objective is that DiSC applies to future secure chip DBMS that involve new hardware, new application requirements, and thus new database techniques. For example, expected hardware evolutions of secure chips are an increase of the communication throughput and CPU power, a much slower increase of the RAM resource and an important evolution of the stable storage technologies (i.e., NOR and NAND FLASH, MEMS, PCM, etc.) [39]. A recent study describes specific storage techniques to manage data on a chip endowed with NOR FLASH [4]. While the hardware constraints differ significantly from EEPROM-based chips, the DiSC metrics capture identically the trade-offs introduced in this study (e.g., resource consumption versus insertion throughput versus query/extraction performance). For instance, the resource consumption metric, which integrates the memory cell size, accommodates well different densities of electronic stable memories. Another important trade-off targets the RAM resource, made critical by its poor density. In [2], we followed a codesign approach to calibrate the RAM resource of a chip to match the performance requirements of embedded applications. This study however did not allow studying several dimensions jointly, a limitation that could be circumvented using DiSC. Some announced secure chips are now endowed with a processor data cache holding up to 1 Kbyte of SDRAM. DiSC metrics allow comparing different techniques and trade-offs on the processor cache, the RAM, and the stable storage competing on the same silicon die. More generally, autonomous secure chips used in novel contexts, such as ambient intelligence, may lead to unusual requirements (e.g., high insertion throughput, low query or extraction performance, aggregation-based authorizations). This motivates the development of new data management techniques that need to be compared thanks to a benchmark.

References

[1] N. Anciaux, C. Bobineau, L. Bouganim, P. Pucheral, and P. Valduriez, "PicoDBMS: Validation and Experience," *Proc. Int'l Conf. Very Large Data Bases (VLDB)*, 2001.

[2] N. Anciaux, L. Bouganim, and P. Pucheral, "Memory Requirements for Query Execution in Highly Constrained Devices," *Proc. Int'l Conf. Very Large Data Bases (VLDB)*, 2003.

[3] D. Bitton, D.J. DeWitt, and C. Turbyfil, "Benchmarking Database Systems: A Systematic Approach," *Proc. Int'l Conf. Very Large Data Bases (VLDB)*, 1983.

[4] C. Bolchini, F. Salice, F. Schreiber, and L. Tanca, "Logical and Physical Design Issues for Smart Card Databases," *J. ACM Trans. Information Systems*, vol. 21, no. 3, 2003.

[5] L. Bouganim, N. Dieu, and P. Pucheral, "MobiDiQ: Mobile Digital Quietude," *Gold Award of Simagine Int'l Contest*, 2005.

[6] M.J. Carey, D. DeWitt, and J. Naughton, "The OO7 Benchmark," *Proc. ACM SIGMOD*, 1993.

[7] R. Cattell and J. Skeen, "Object Operations Benchmark," *J. ACM Trans. Database Systems*, vol. 17, no. 1, 1992.

[8] Common Criteria, "Common Criteria for Information Technology Security Evaluation Part 1," CCIMB-2005-08-001, 2005.

[9] E. Damiani, S. De Capitani Di Vermicati, S. Paraboschi, and P. Samarati, "A Fine-Grained Access Control System for XML Documents," *J. ACM Trans. Information and System Security*, vol. 5, no. 2, 2002.

[10] J.G. Dyer, M. Lindemann, R. Perez, R. Sailer, L. Doorn, and S.W. Smith, "Building the IBM 4758 Secure Coprocessor," *Computer*, vol. 34, no. 10, Oct. 2001.

[11] B. Finance, S. Medjoub, and P. Pucheral, "The Case for Access Control on XML Relationships," *Proc. ACM Int'l Conf. Information and Knowledge Management (CIKM),* 2005.

[12] G. Graefe, "Query Evaluation Techniques for Large Databases," *J. ACM Computing Surveys,* vol. 25, no. 2, 1993.

[13] J. Gray, *The Benchmark Handbook for Database and Transaction Processing Systems.* Morgan Kaufmann, 1991.

[14] R. Gupta, S. Dey, and P. Marwedel, "Embedded System Design and Validation: Building Systems from IC Cores to Chips," *Proc. Int'l Conf. VLSI Design (VLSI),* 1998.

[15] J. Karlsson, A. Lal, C. Leung, and T. Pham, "IBM DB2 Everyplace: A Small Footprint Relational Database System," *Proc. IEEE Int'l Conf. Data Eng. (ICDE),* 2001.

[16] W. Kim, "Smart Cards: Status, Issues, US Adoption," *J. Object Technology,* vol. 3, no. 5, 2004.

[17] D.L. Maloney, "Card Technology in Healthcare," *Proc. CardTech SecurTech (CTST),* 2001.

[18] Mastercard, *MasterCard Open Data Storage (MODS),* 2002.

[19] M. Missikoff and M. Scholl, "Relational Queries in a Domain Based DBMS," *Proc. ACM SIGMOD,* 1983.

[20] ISO/IEC 21000-5-2004 Standard, *MPEG-21 Right Expression Language (MPEG-REL),* 2004.

[21] *Open Digital Rights Language Initiative,* www.odrl.net, 2006.

[22] Oracle Corporation, *Oracle 9i lite: Release Notes—Release 5.0.1,* 2002.

[23] O. Potonniée, "A Decentralized Privacy-Enabling TV Personalization Framework," *Proc. European Conf. Interactive Television (EuroITV),* 2004.

[24] P. Pucheral, L. Bouganim, P. Valduriez, and C. Bobineau, "PicoDBMS: Scaling Down Database Techniques for the Smart Card," *J. Very Large Data Bases,* vol. 10, nos. 2-3, 2001.

[25] P. Pucheral, J.-M. Thevenin, and P. Valduriez, "Efficient Main Memory Data Management Using the DBGraph Storage Model," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* 1990.

[26] *SAP Standard Application Benchmarks,* www.sap.com, 2006.

[27] A.R. Schmidt, F. Waas, M.L. Kersten, M.J. Carey, I. Manolescu, and R. Busse, "XMark: A Benchmark for XML Data Management," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* 2002.

[28] B. Schneier and A. Shostack, "Breaking Up Is Hard to Do: Modeling Security Threats for Smart Cards," *Proc. Usenix Symp. Smart Cards,* 1999.

[29] P. Seshadri and P. Garrett, "SQLServer for Windows CE-A Database Engine for Mobile and Embedded Platforms," *Proc. IEEE Int'l Conf. Data Eng. (ICDE),* 2000.

[30] Smart Card Alliance, *HIPAA Compliance and Smart Cards: Solutions to Privacy and Security Requirements,* www.smartcardalliance.org, 2003.

[31] *The SmartRight Content Protection System,* www.smartright.org, 2006.

[32] Sybase Inc., *The Next Generation Database for Embedded Systems,* white paper, 2000.

[33] Trusted Computing Group, www.trustedcomputing.org, 2006.

[34] *Transaction Processing Performance Council,* www.tpc.org, 2006.

[35] *TPC Benchmark H,* www.tpc.org/tpch/spec/tpch2.3.0.pdf, 2006.

[36] P. Valduriez, "Join Indices," *ACM Trans. Database Systems,* vol. 12, no. 2, 1987.

[37] H. Vogt, M. Rohs, and R. Kilian-Kehr, *Middleware for Communications,* chapter 16. John Wiley & Sons, 2003.

[38] *XrML eXtendible Rights Markup Language,* www.xrml.org, 2006.

[39] H. Yu, D. Agrawal, and A. El Abbadi, "Tabular Placement of Relational Data on MEMS-Based Storage Devices," *Proc. Int'l Conf. Very Large Data Bases (VLDB),* 2003.

Nicolas Anciaux received the PhD degree in computer science from the University of Versailles in 2004. He is a researcher at INRIA, France. After one year at the University of Twente, The Netherlands, he joined the Secured and Mobile Information Systems (SMIS) team, INRIA Rocquencourt. His main research area is embedded data management and database security and privacy, in particular, in the context of ambient intelligence. He has coauthored about 15 conference and journal papers.

Luc Bouganim received the PhD degree and the Habilitation à Diriger des Recherches from the University of Versailles in 1996 and 2006, respectively. He is a director of research at INRIA Rocquencourt. He worked as an assistant professor from 1997 to 2002 when he joined INRIA. He is the coauthor of more than 60 conference and journal papers and an international patent. He was the recipient of three international awards. Since 2000, he has been strongly engaged in research activities on ubiquitous data management and data confidentiality. He is currently the vicehead of the Secured and Mobile Information Systems (SMIS) research team.

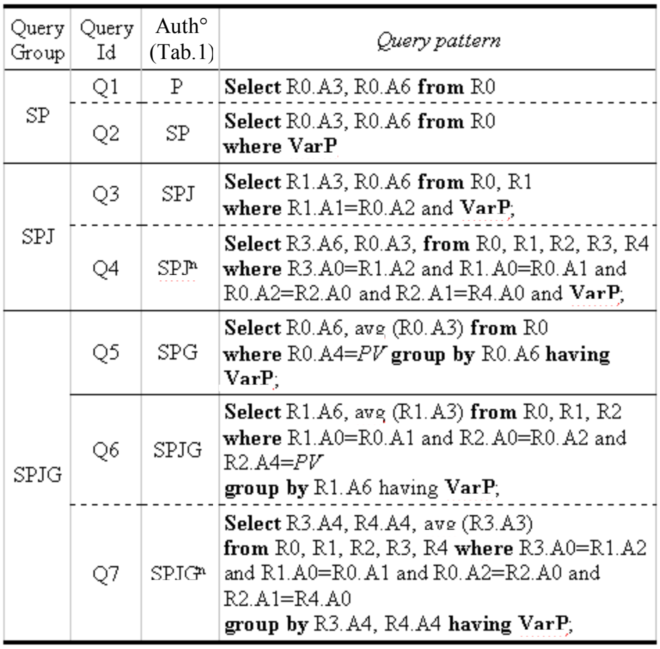
Philippe Pucheral received the PhD degree in computer science from the University Paris 6 in 1991 and the Habilitation à Diriger des Recherches from the University of Versailles in 1999. He is a professor at the University of Versailles, currently in secondment at INRIA Rocquencourt, where he is heading the Secured and Mobile Information Systems (SMIS) research team. He is the author or coauthor of more than 60 conference and journal papers, four international patents, and four books. He was the recipient of four international awards (EDBT '92, VLDB '00, e-gate '04, SIMagine '05). His domain of interest covers database systems, mobile and embedded databases, and database security.

Patrick Valduriez received the PhD degree and Doctorat d'Etat in computer science from the University Paris 6 in 1981 and 1985, respectively. He is a director of research at INRIA, France, and the manager of the Atlas research team in Nantes, pursuing research in data management in distributed systems. He is the author or coauthor of more than 170 technical papers and several books, among which are *Principles of Distributed Database Systems* (1991 and 1999), *Object Technology* (1997), and *Relational Databases and Knowledge Bases* (1990). He has been a trustee of the VLDB endowment and an associate editor for several journals, including *ACM Transactions on Database Systems*, the *VLDB Journal*, *Distributed and Parallel Databases* (DAPD), *Internet and Databases*, *Web Information Systems*, etc. He was the general chair of SIGMOD/PODS 2004 in Paris and is the general chair of EDBT 2008 in Nantes. He was the recipient of the 1993 IBM France scientific prize. He obtained the best paper award at the VLDB 2000 conference.

APPENDIX A: DISC QUERY SET

DiSC contains seven query patterns (Q1 to Q7) organized in three query groups (SP, SPJ and SPJG). Query patterns include a variable predicate (VarP) to vary the selectivity as well as the number and location (in the join path) of selection predicates. As mentioned in Section 3.3, query selectivity always refers to selection selectivity. Indeed, since the data is generated randomly following a uniform distribution (see appendix B), applying a selection on any relation involved in a Select-Project-Join query reduces to the same extent the query result size. The query set should allow for selectivity of 90% (extraction of an authorized view), 10 % and 1% (to reflect regular queries on authorized views) and 0% (to reflect an unsuccessful search). To this end, selection predicates apply to attributes A4 and A5 varying on fixed size domains of cardinality 10 and 100 respectively. PresentValue or PV for short (resp. MissingValue or MV), denotes a string value that exists (resp. does not exist) in the domain under consideration. Following this convention, VarP predicates (A4 ≠ PV), (A4 = PV), (A5 = PV) and (A5 = MV) are used to enforce selectivities of 90%, 10%, 1% and 0% respectively. The query set is depicted in Table 6.

TABLE 6
DISC QUERY SET



Query Group	Query Id	Auth° (Tab.1)	*Query pattern*
SP	Q1	P	**Select** R0.A3, R0.A6 **from** R0
	Q2	SP	**Select** R0.A3, R0.A6 **from** R0 **where VarP**
SPJ	Q3	SPJ	**Select** R1.A3, R0.A6 **from** R0, R1 **where** R1.A1=R0.A2 and **VarP**;
	Q4	SPJ^n	**Select** R3.A6, R0.A3, **from** R0, R1, R2, R3, R4 **where** R3.A0=R1.A2 and R1.A0=R0.A1 and R0.A2=R2.A0 and R2.A1=R4.A0 and **VarP**;
SPJG	Q5	SPG	**Select** R0.A6, avg (R0.A3) **from** R0 **where** R0.A4=*PV* **group by** R0.A6 **having** **VarP**;
	Q6	SPJG	**Select** R1.A6, avg (R1.A3) **from** R0, R1, R2 **where** R1.A0=R0.A1 and R2.A0=R0.A2 and R2.A4=*PV* **group by** R1.A6 having **VarP**;
	Q7	SPJG^n	**Select** R3.A4, R4.A4, avg (R3.A3) **from** R0, R1, R2, R3, R4 **where** R3.A0=R1.A2 and R1.A0=R0.A1 and R0.A2=R2.A0 and R2.A1=R4.A0 **group by** R3.A4, R4.A4 **having VarP**;

The SP query group includes query patterns Q1 and Q2:

- Q1 (P) is self-explanatory.
- Q2 (SP) is a mono-attribute selection on relation R0 with varying selectivity.

The SPJ query group includes query patterns Q3 and Q4:

- Q3 (SPJ) is a mono-attribute selection on two joining relations R0 and R1 with varying selectivity. Two variations of Q3 are studied, locating the selection on either R0 or R1 to induce key→foreign key or foreign key→ key traversals.
- Q4 (SPJn) joins the five relations of the dataset (R0 to R4) and applies a mono- or multi-attribute selection. Similar to Q3, mono-attribute selections apply either on R0 or R1. Multi-attribute selections are applied on R3 and R4 and lead to slightly different query selectivities; i.e., 81% (R3.A4 ≠ PV and R4.A4 ≠ PV) for extraction, and 9% (R3.A4 = PV and R4.A4 ≠ PV), 1% (R3.A4 = PV and R4.A4 = PV) and 0% (R3.A5 = MV and R4.A5 = MV) for regular queries.

Finally, the SPJG query group includes query patterns Q5, Q6 and Q7:

- Q5 (SPG) computes the average value of attribute R0.A3 for each R0.A6 group. VarP implements here a having clause, in order to apply a selectivity on the aggregate results of 90% (min(R0.A3) ≤ threshold90) for extraction queries, 10% (min(R0.A3) ≤ threshold10) and 0% (min(R0.A3)<0) for regular queries. Annex B presents the dataset generation and make precise the computation of threshold90 and threshold10.
- Q6 (SPJG) is built similarly to Q5 with additional joins between relations R0, R1 and R2. VarP implements a having clause involving min(R1.A3).
- Q7 (SPJGn) joins all relations and performs a multi-attribute aggregation, producing 100 distinct values for (R3.A4, R4.A4) group. VarP implements a having clause involving min(R3.A3).

APPENDIX B: DATA SET GENERATOR

The data set generated must comply with the data set description given in Section 3.2 and must exhibit the attribute selectivity expected by the queries presented in appendix A. While planned selectivity are easy to guarantee for a regular selection, the problem is more difficult for selectivity of a having clause, which depends on aggregated values. Let us consider Q5 with 10% selectivity: Select A6, avg (A3) from R0 where A4=PV group by A6 having min (A3) ≤ threshold10. Generating attributes A3 and A6 of R0 randomly and independently would lead roughly to the same average, minimum and maximum value for all A6 groups, thus making threshold10 impossible to fix. The solution proposed is to derive A6 values from A3 values (denoted by A3→A6) in such a way that, for A3 having d distinct numeric values in the range [1..d], threshold10 = d/10. Thus, the planned selectivity for Q5, Q6 and Q7 can be obtained by enforcing respectively R0.A3→R0.A6, R1.A3→R1.A6, R3.A3→R3.A4 at data generation time.

To make the data generation reproducible, we use a pseudorandom number generator initialized with a given seed. As explained in Section 3.2, we do not consider skewed data distribution. The pseudorandom function PR used takes as input a positive integer n greater than zero, and delivers a uniformly distributed pseudorandom positive integer p such that p ∈ [1..n].

For N tuples generated in R0, N/6 tuples must be generated in R1 and R2, and N/36 tuples in R3 and R4. The data set generation occurs as follows. An array of character strings, denoted by STR[ ], is first built using PR with an average size of 18 characters. For attribute Ri.Aj, each generated string S is concatenated with indices ij to get a 20 characters string. The size of STR[ ] is bound to N/10, the largest number of distinct values for an attribute (attribute A6 of R0).

The database is populated one relation after the other, following an order fixed by the referential integrity constraints (e.g., R4, R3, R2, R1, R0). Let us detail the generation of R0. R0 contains 7 attributes named A0 to A6. A0, the primary key, is generated sequentially from 1 to N. Foreign key A1 (resp. A2) references one of the N/6 tuples of R1 (resp. R2). To get a uniform distribution, pseudorandom numbers are picked between 1 and N/6 using PR. A3 is a numeric attribute populated by picking pseudorandom numbers between 1 and N/10 using PR. A4 (resp. A5) is a string attribute having 10 (resp. 100) distinct values. Thus, A4 values are set to STR[PR(10)] concatenated with the string "04" (meaning relation R0, attribute A4). Similarly, A5 values are set to STR[PR(100)] || "05". Finally, A6 values are set to STR[A3] || "06" to enforce A3→A6.