Future Trends in Secure Chip Data Management

Nicolas Anciaux, Luc Bouganim and Philippe Pucheral
INRIA Rocquencourt and PRiSM Laboratory, University of Versailles, France
{firstname.lastname}@inria.fr

Abstract

Secure chips, e.g. present in smart cards, TPM, USB dongles are now ubiquitous in applications with strong security requirements. Secure chips host personal data that must be carefully managed and protected, thus requiring embedded data management techniques. However, secure chips have severe hardware constraints which make traditional database techniques irrelevant. We previously addressed the problem of scaling down database techniques for the smart card and proposed the design of a DBMS kernel called PicoDBMS. This paper summarizes the learning obtained during an extensive performance analysis of PicoDBMS. Then it studies to which extent secure chips hardware evolution should impact the design of embedded data management techniques. Finally, it draws research perspectives linked to a broader usage of secure chip data management techniques.

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, Trusted Platform Modules (TPM [15]) and other forms of radio-frequency or pluggable smart tokens like USB dongles [7] or Mass Memory Cards [13]. Smart card is the most popular secure chip form factor and is used worldwide in applications such as banking, pay-TV, GSM subscriber identification, loyalty, healthcare and transportation. Now, large-scale governmental projects are pushing for an ever wider acceptance of smart cards (passport, driving license, e-voting, insurance or transport). Secure chips are also at the heart of computing systems, to protect PC platforms against piracy [15] or to enforce Digital Right Management in rendering devices [14]. Finally, chips are today integrated in a large diversity of usual objects to form an ambient intelligence surrounding. While security seems not the primary concern in this latter case, the strong demand of individual for enforcing their elementary rights to privacy quickly changes the situation.

In all these scenarios, data hosted by the secure chips is personal and must be carefully managed and protected. Embedded data management techniques are thus required to store the data securely (thanks to the chip tamper resistance) and act as a trusted doorkeeper, which authenticates each user (i.e., human being or software) and solely delivers her authorized view of the data. While a full fledged on board DBMS is not always required in all scenarios, access control policies may entail on-board computation of select, join and aggregate operators. Secure chips have however strong hardware constraints, and more importantly a very unique resource balance (e.g., powerful CPU vs. tiny RAM, very fast read vs. very low write in stable storage, etc) which make current database techniques, even those designed for lightweight DBMS [11] irrelevant.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

In a previous paper, we addressed the problem of scaling down database techniques for the smart card and proposed the design of a DBMS kernel called PicoDBMS [12]. We recently conduct extensive performance measurements of PicoDBMS on a Gemalto's smart card experimental platform and on a hardware cycle-accurate simulator allowing conducting tests on databases exceeding the smart card storage capacity. The objective was to assess PicoDBMS overall performance as well as the relative performance of candidate storage and indexing data structures.

The first contribution of this paper is to expose the learning obtained during this long and tricky experimentation process thus answering the question "an the PicoDBMS design effectively accommodate real hardware platforms?". Since the initial PicoDBMS design, secure chip hardware evolved following roughly Moore's law. A second question is then "Is scaling down database techniques for secure chips still a topical issue today?". The second contribution of this paper is to study the hardware progress and show how it impacts the design of data management techniques, opening interesting research issues. Finally, "Is PicoDBMS the definite answer to a sporadic problem or rather a first step towards a broader and longer term research?" is a legitimate question putting database management on secured chip in perspective. If PicoDBMS has proved to be an effective answer to the initial problem statement (i.e., on-chip DBMS), it does not answer all the situations where data management could benefit from secured chips. To illustrate this, chip manufacturers are announcing Mass Storage Devices [7, 10, 13] combining a secured chip with a huge amount (several GB) of unsecured external stable memory, dictating revisiting again the database technology on chip. The third contribution of the paper is to devise new research perspectives considering extending the sphere of confidentiality of secure chips toward external (unsecured) resources.

The paper is organized in three sections (Sections 2-4) addressing the three contributions mentioned above, followed by a conclusion (Section 5).

2 PicoDBMS design and evaluation

Current powerful smart cards include in a monolithic chip, a 32 bit RISC processor clocked at about 50 MHz, memory modules composed of about 96 KB of ROM, 4 KB of static RAM and 64 KB of EEPROM, 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). EEPROM is used as stable memory to store persistent information, and holds data and downloaded programs (in case of multi-application cards).

The smart card internal resource balance is very unique. Indeed, the on-board processing power, calibrated to sustain cryptographic computations and enforce smart cards security properties, is oversized against the small amount of embedded data. In addition, EEPROM shares strong commonality with RAM in terms of granularity (direct access to any memory word) and read performance (60-100 ns/word), but suffers from a dramatically slow write time (about 10 ms per word). Finally, only a few bytes of RAM remain available for the embedded applications, the major part of the RAM being preempted by the operating system and the virtual machine (in case of Java enabled cards).

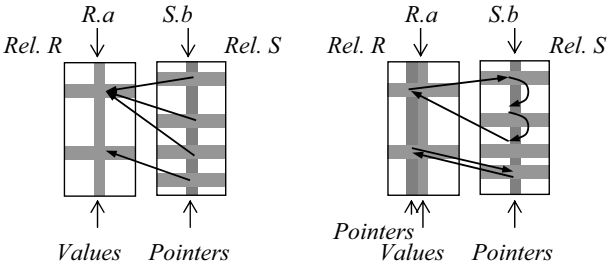
As mentioned above, PicoDBMS must act as a trusted doorkeeper delivering to each authenticated user authorized view of the data. A trivial solution would be to materialize each view in a separate file. However, this badly adapts to data and access control policies evolutions and would result in a huge data replication among files, thereby hurting the smart card limited storage constraint. Thus, a dynamic evaluation of the authorized views must be considered, leading to embed on chip the ability to compute database operators and run query execution plans implementing views. The database components that must be embedded are thus: a storage manager to organize data and indices 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 data confidentiality and can be executed off card.

To match the smart card aforementioned hardware constraints, we have defined a set of design rules for on-chip database components [12]: *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 over-dimensioned CPU) and *Security rule* (never externalize private data, minimize code complexity). For the sake of conciseness, we recall below the storage, indexation and query execution techniques compliant with those rules and summarize performance results from [3]. The complete design of PicoDBMS is described in [12].

2.1 Storage and indexing models

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 access locality. However, FS is space consuming (all duplicate attribute values are stored) and inefficient (without index structures, FS relies on sequential scans for all operations). Since locality is no longer an issue in our context (Read and Access rules), pointer-based storage models 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). 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 their FS counterpart, their implementation is more efficient in a smart card because the amount of data to be written is smaller (Write rule).

Let us now consider index compactness. A select index is typically made of a collection of values and a collection of pointers linking each value to all tuples sharing it (named value-to-tuple pointers). 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, we store these pointers 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. 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 tuples 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).



a. Domain Storage. b. Ring Storage.

Figure 1: Storage models for foreign keys.

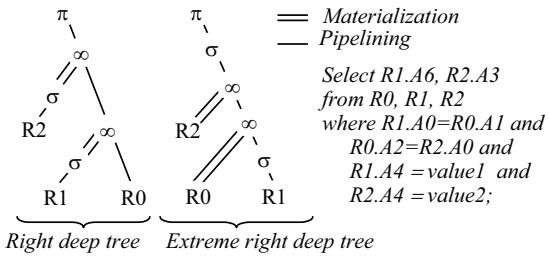


Figure 2: Extreme right deep tree example.

Join indices can be treated in a similar way. A join predicate of the form (*R.a=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 (Fig. 1a). Traversals from R.a to S.b can be optimized too by RS, a bi-directional join index being obtained by a ring index on S.b (Fig. 1b).

2.2 Query processing

Devising a query execution engine compliant with the design rules introduced above is challenging. Indeed, traditional query processing strives to exploit large main memory for storing temporary data structures (e.g., hash tables) and intermediate results and resorts to materialization on disk in case of memory overflow, which hurts both Read and Write rules. To address this issue, we propose query processing techniques that do not use any working RAM area (except for a small collection of cursors) nor incur any writes in stable memory.

Let us consider the execution of SPJ (Select-Project-Join) queries. PicoDBMS combines all operators in an extreme right-deep tree execution plan, which leads to pure pipeline execution without materialization. Fig. 2 compares a classical plan with an extreme right deep tree on a SPJ query. 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. A query execution plan is activated starting at the root of the operator tree. The dataflow 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 predicate. Depending on the storage model, attribute values are directly read in the tuple (FS) and/or reached by dereferencing a pointer (DS) and/or by following a pointers ring (RS). With RS, the selection predicate (or part of it whether multi-attribute) can be evaluated on the distinct values of a domain, and the matching tuples are directly retrieved 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) and/or following the pointers ring (RS) to reach the value present in the input tuple. 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 indices (DS/RS), the cost of joins depends on the way indices 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 $n \times m$ starting with R. With RS, the join cost becomes proportional to $n + m$ starting with R and to $m^2/2n$ starting with S (retrieving the R tuples associated to each S tuple incurs traversing half of a ring in average).

Let us finally consider the execution of the aggregate operator (sort is not described since it can be performed off card). At first glance, pipeline execution is not compatible with aggregation, typically performed on materialized intermediate results. Our solution exploits two properties: (i) aggregate can be done in pipeline if the incoming tuples are yet grouped by distinct values and (ii) 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 has to be rearranged. Multi-attribute aggregation is trickier to implement since it imposes one (resp. several) Cartesian product operator at the leaf of the tree in order to produce tuples ordered by a couple (resp. tuple) of distinct grouping values.

2.3 PicoDBMS performance

Two different platforms, provided by our industrial partner Gemalto, have been used to measure PicoDBMS performance: a real smart card prototype compliant with the aforementioned hardware characteristics and a cycle-accurate hardware simulator of this prototype. The hardware simulator allows considering datasets exceeding the smart card storage capacity, up to 1MB. It delivers the exact number of CPU cycles between two breakpoints set in the embedded code and thus provides exact performance predictions.

Several versions of PicoDBMS have been measured: *P-FS* (PicoDBMS-FS version) supports only the FS storage model; *P-DS* is enhanced with the DS model for redundant attributes and foreign keys serving as a unidirectional join index as shown in Fig. 1a; *P-RS* also supports the RS model both for selection and bidirectional join indices. We evaluate the performance on a synthetic dataset composed of 5 joining tables, varying the number of embedded tuples (up to 50000 tuples), with simple (SP: select-project), regular (SPJ: select-project-join) and complex queries (SPJG: select-project-join-aggregate). We refer the reader to [3] for further details on the performance evaluation process and summarize below the main results regarding storage consumption, insertion and query performance.

Stable storage consumption depends both on the database and the DBMS code footprint. While code footprint reduction has been exploited as a marketing advantage by lightweight DBMS vendors, 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 footprints of *P-FS* (26 KB) and *P-RS* (39 KB). The database footprint is then the dominant factor except for very small datasets. As expected, *P-FS* is the least compact since it does not benefit from any form of compression; *P-DS* is the most compact, domains acting as a dictionary compression scheme (78% saving in our dataset); and the extra storage consumption incurred by *P-RS* is rather small (13% worse than *P-DS*), highlighting the high compactness of ring indices.

Insertion throughput is acceptable whatever the DBMS version. Insert performance is essentially impacted by the amount of writes in EEPROM. To this respect, *P-DS* performs better than *P-RS* itself performing better than *P-FS* roughly in the proportions mentioned above for the database footprint. Note that the tuple creation cost and the logging cost are independent of the relation size while integrity checking and domain update depend on the relation and domain cardinality, respectively. This balance the cost of updating indexing structures with the benefit provided to check integrity constraints.

Query execution performance, measured in throughput (result tuples produced per second), depends on the query complexity and on the presence of indexing structures. Simple queries (SP) reach high throughput delivery for any DBMS version (thousands of tuples/s). *P-RS* uses rings as selection indices, thus offering the highest throughput for queries with high to medium selectivity but looses its advantage on *P-FS* and k for queries with very low selectivity or no selection (projection overhead). Regular queries (SPJ) need *P-DS* or k to reach a good performance (hundreds of tuples/s). Indeed, *P-FS* yields low throughput because joins are performed by nested loops. With *P-DS*, the DS model implements a unidirectional join index imposing a unique join ordering in the query execution plan (e.g., from R to S on Fig. 1a) and precluding applying some selections first. *P-RS* takes advantage of the bidirectional join index provided by RS (see Fig. 1b), thus performing almost one order of magnitude better than *P-DS*. Complex queries (SPJG) require *P-RS* to fairly support aggregation (tens to a hundred of tuples/s). The important difference between SPJ and SPJG queries is explained by the high difficulty to handle aggregations without consuming RAM. Thanks to rings, joins within aggregation queries are handled with a limited degradation. The throughput remains stable when increasing the database size for aggregations on a single attribute. For multi-attribute aggregations, the ring index can help for only one of them, the others requiring Cartesian products, making the performance dependent on the database size.

The conclusion of this performance analysis is threefold. First, the performance results show without ambiguity that the PicoDBMS design (more precisely *P-RS*) effectively accommodates real hardware platforms thus answering positively the first question stated in the introduction. Second, the comparison between the PicoDBMS versions provides hints to select the most appropriate storage and indexing model for a given application. *P-FS* may suit very simple applications (small database, simple queries) if code simplicity is a main requirement. Note however that this intuitive choice should be reconsidered if database footprint or insertion cost is an important concern. *P-DS* sustains regular queries on large datasets, and *P-RS* fulfils the requirements of more complex applications involving aggregate queries. Third, this study highlighted two fundamental differences between traditional DBMS and on chip DBMS evaluation: (i) while performance is the usual metrics for traditional DBMS, no single metrics dominates when evaluating on chip DBMS (e.g., performance, code simplicity, storage consumption); (ii) the determination of on chip DBMS performance limits (e.g., maximal

number of on-board tuples satisfying storage constraints or query performance requirements) is crucial and may help fixing co-design hints, considering that the hardware platform is not adjustable after manufacturing. We are currently working on a benchmark for on-chip DBMS taking these specificities into account.

3 Research Perspectives Driven by Hardware Properties

To answer the second question raised in the introduction, that is whether scaling down database techniques for secure chips is still a topical issue, we analyze the main foreseeable hardware trends for secure chips an discuss how they impact the design of embedded data management techniques.

3.1 Secure chips hardware trends

CPU resource: during the last ten 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. At least two factors advocate for a continuous growth of CPU power. First, the rapid evolution of secure chip communication throughput (e.g., high delivery contactless cards, USB cards) allows secure chip applications to evolve towards CPU intensive data flow processing using cryptographic capabilities (e.g., DRM). Second, many efforts from secure chip manufacturers focus on multi-applications and multi-threaded operating systems demanding even more CPU power. In addition, note that increasing the CPU power as little impact on the silicon die size, an important parameter in terms of tamper-resistance and production cost on large scale markets.

RAM resource: secure chips hold today a few KB of static RAM, almost entirely consumed by the operating system (and the Java virtual machine in Java enabled chips). The gap between the RAM left available to the applications on one side and the CPU and stable storage resources on the other side will certainly keep on increasing. 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, 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. RAM competing with stable memory on the same die, secure chip manufacturers favor the latter against the former to increase the chip storage capacity, thus enlarging their application scope.

Stable storage resource: secure chips rely on a well established and slightly out-of-date hardware technology (0.35 micron) to minimize production costs. However, the market pressure generated by emerging applications leads to a rapid increase of the storage capacity. Taking advantage of a 0.18 micron technology allows doubling the storage capacity of EEPROM memories. At the same time, manufacturers are integrating 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 (holding commonly 64 pages). In a long term perspective, researchers in memory technologies aim at developing highly compact non-volatile memories providing fast read/write operations with fine grain access (e.g., MEMS, PCM, etc.). But integrating these future technologies in a secured chip introduces intrinsic difficulties: very high security level that needs to be proved for any new technology, low manufacturing cost motivating the usage of old amortized technologies, complexity to integrate all components in the same silicon die.

To conclude, PicoDBMS has been designed a couple of years ago to tackle the rather unusual computing environment provided by secure chips. This environment was characterised by the three following properties: (P1) High processing power wrt the amount of RAM and on-chip data; (P2) Tiny RAM wrt the amount of on-chip data; (P3) Fast reads but slow writes in stable storage. As discussed above, these properties are expected to remain stable in the future, at least until the integration of emerging memory technologies like MEMS and

PCM. However, the diversification of hardware solutions (e.g., alternatives to EEPROM now exist) opens up new perspectives to tackle the secure chip constraints.

3.2 Hardware driven research issues

Stable storage technology: FLASH memory is becoming an effective alternative for secured chips due to its compactness, leading to new secure chip memory architectures mixing EEPROM, NAND and NOR FLASH. The different memory properties should be taken into account when designing embedded database components and specifically when designing the data storage and indexing models. [4] proposes techniques to store efficiently data on a smart card equipped with NOR FLASH but is limited to mono relation queries. NAND FLASH advocates for a sequential storage model (coarse grain erasure) while the reduced RAM of secured chips advocates for a highly indexed storage model. Hybrid chip architectures (e.g., mixing EEPROM and NAND FLASH) may lead to partition the data according to its update frequency/complexity (e.g., indices in EEPROM and data in FLASH). A co-design study would be particularly helpful to determine the best balance between the different types of memory. Long term memory alternatives could also be envisioned, including technologies like PCM, OUM and Millipedes. There are already some works on Millipedes [16] in a classical database context. Combining the memory properties with the other hardware constraints will undoubtedly generate important problems.

RAM: RAM is a crucial resource for the evaluation of database queries. PicoDBMS has been designed to cope with this issue (RAM rule), relying on an intensive use of indices. While this solution is convenient for the targeted context, indices may be disqualified in update intensive contexts (e.g., sensed data) because they increase update cost (especially in FLASH memory), they make update atomicity more complex to implement and they competes with on-board data. In addition, throwing away indices may help reducing the database footprint and the code complexity in contexts where performance is not the major concern. Thus, we consider that designing RAM constrained query execution techniques will remain a hot topic in the future. In [2] we proposed a first solution to minimize the RAM consumption in the query execution, assuming no index. First, we designed a RAM lower bound query execution mode inducing several iterations on the data to compute the query result. Second, we proposed a new form of optimization, called iteration filter, which drastically reduces the cost incurred by the preceding model, without hurting this RAM lower bound. Finally, we analyzed how to benefit from an incremental growth of the RAM amount. This work helps to determine: how much RAM should be added to reach a given response time, how much the expected response time should be relaxed to tackle a given query with a given quantity of RAM, how much data can be embedded in a given device without hurting an expected execution time. This is a first step towards more complete co-design work aiming at calibrating all resources of a hardware platform.

Cache conscious strategies: Another interesting research issue is improving the processor data cache usage when processing queries. Indeed, some announced smart cards are now endowed with a cache holding up to 1 KB. Since query execution with a constrained RAM induces numerous iterations on the data (e.g., to perform group by, joins without index), cache conscious algorithms could bring great performance improvements, diminishing the overheads of these iterations. Cache conscious strategies have been envisioned on traditional and main-memory DBMS and led to reorganize indices storage and data layout [1]. The challenge is that cache conscious processing generally leads to rethink data and index storage organization in a way which may contradict other concerns like data compactness and specific storage models dedicated to new stable memories. Co-design could again be helpful to calibrate the processor cache and the RAM, both competing on the same silicon die.

Contactless interactions: Contactless interfaces are more and more promoted by constructors and governmental organizations for their ease of use. While the current PicoDBMS design could adapt contactless smart cards, whether the processing requirements (in terms of query types and execution time) remain the same is an important question. For instance, severe response time constraints have to be enforced to avoid the card holder to stop when passing near a contactless reader. A possible solution could be to reduce the completeness/accuracy of the result for applications accepting partial results (e.g., top queries, approximate answers).

4 Extending the Sphere of Confidentiality to External Resources

Secured chips are often plugged into or linked to more powerful devices (e.g., cell phone, PDA, PC, and even servers). Thus, it does make sense to take advantage of the device resources to overcome the secured chip limitations. The idea is to extend the sphere of confidentiality provided by the security properties of the chip to external resources. This is actually the approach followed by the TPM [15] architecture where the secured chip protects the whole PC platform. This section sketches how to extend the sphere of confidentiality to external (unsecured) storage resources, to external processing resources and finally to a shared database server.

External Storage Resources: One of the main limiting factors of secure chips to address new domains of applications is the tiny capacity of the stable memory. To tackle this issue, smart card manufacturers are pushing new architectures combining in the same form factor (e.g., USB dongle) a smart-card-like secure microcontroller with an Gigabyte sized external (then unsecured) FLASH memory module (e.g., Gemalto's SUMO "smart card", Renesas X-Mobile Card [13]). To prevent from information disclosure and protect the integrity of the data stored on the unsecured external memory, cryptographic techniques (e.g., encryption, secure hash functions, etc.) must be employed. Some metadata (encryption keys, checksums, freshness information, bootstrap, etc.) and the query evaluator itself must remain embedded in the secured chip to deliver only the authorized data to the user. The problem is to combine cryptographic techniques and query execution techniques in a way satisfying three conflicting objectives: efficiency, high security and compliance with the chip hardware resources.

External Processing Resources: External computing resources (e.g., the CPU and RAM of the terminal the secure chip in connected with) could be exploited by delegating some expensive computation (e.g., sorting, duplicate removal). However, this incurs two complex problems. First, the correctness of the delegated computation must be checked, and if some techniques exist for simple selections [8], no satisfactory solution has been proposed so far for more complex operations. Second, delegating computation should not reveal any sensitive information. Since some information can be inferred even from encrypted data, this remains a challenging issue.

External Shared Database: Up to now, we considered that data sharing is under the control of a single chip, the external resources being considered as direct extensions of this chip. Assuming the database be stored on a remote and unsecured server and be shared among different users, the spectrum of applications grows. For instance, one may consider a shared database hosted by an — untrusted — Database Service Provider. The Database Service Provider Model has been studied for the first time in [9], but sharing was not a concern in this study. The sharing issue has been tackled in [5] thanks to an architecture called Chip-Secured Data Access (C-SDA). C-SDA is a client-based security solution where a smart card acts as an incorruptible mediator between a user and a remote encrypted database. The smart card checks the user's privileges, participates in the query evaluation and decrypts the final result before delivering it to the user. Since then, this hot topic generated several papers including [6]. So far, existing studies focused only on the confidentiality of the remote data but disregarded tamper-resistance (including opacity, integrity and freshness). Solutions relying on a secured co-processor at the server side could also be investigated and would raise new research challenges.

5 Conclusion

Secure chips have severe hardware constraints which have triggered a deep revisiting of traditional database techniques, with unusual design choices, leading to build a DBMS kernel called PicoDBMS. The objective of this paper was to answer three important questions. The first question was related to the feasibility of the PicoDBMS approach. We showed that the joint efforts from our team and from our industrial partner Gemalto led to a convincing prototype which effectively accommodates real hardware platforms. The second question was whether the problem addressed in the PicoDBMS study is still a topical issue despite the hardware progress. We analyzed the secure chips hardware trends and showed that many important research issues still need be explored. Finally, the third question was whether PicoDBMS opens up a broad and long term research. We

showed that secured chip can be integrated in an insecure distributed computing system (e.g., a database system) and be the ultimate trusted party the complete security relies on. We sketched some research perspectives in this promising direction. Hence, we expect that this paper will contribute to the definition of an exciting research agenda for database management on secure chip.

References

[1] A. Ailamaki, D.J. DeWitt, M.D. Hill: Data Page Layouts for Relational Databases on Deep Memory Hierarchies. In Proc. of International Conference on Very Large Databases, 2002.

[2] N. Anciaux, L. Bouganim, P. Pucheral: Memory Requirements for Query Execution in Highly Constrained Devices. In Proc. of International Conference on Very Large Databases, 2003.

[3] N. Anciaux, L. Bouganim, P. Pucheral: Smart card DBMS: where are we now? INRIA technical report 80840, 2006.

[4] C. Bolchini, F. Salice, F. Schreiber, L. Tanca: Logical and Physical Design Issues for Smart Card Databases. *In Journal of ACM TOIS*, 21(3), 2003.

[5] L. Bouganim, P. Pucheral: Chip-Secured Data Access: Confidential Data on Untrusted Servers. In Proc. of International Conference on Very Large Databases, 2002.

[6] E. Damiani, S. De Capitani Vimercati, S. Jajodia, S. Paraboschi, P. Samarati: Balancing Confidentiality and Efficiency in Untrusted Relational DBMSs. In Proc. of ACM Conference on Computer and Communications Security, 2003.

[7] Dekart SRL.: Dekart Smart Container, 2007. http://www.dekart.com/products/integrated/smart_container

[8] M. Gertz, A. Kwong, C. Martel, G. Nuckolls, P. Devanbu, S. Stubblebine: Databases that tell the Truth: Authentic Data Publication. *Bulletin of the Technical Committee on Data Engineering*, 2004.

[9] H. Hacigumus, B. Iyer, C. Li, S. Mehrotra: Executing SQL over Encrypted Data in the Database-Service-Provider Model. In Proc. of ACM International Conference on Management of Data, 2002.

[10] L. Hamid: New directions for removable USB mass storage, Press release, 2006. http://www.itwales.com/997893.htm.

[11] A. Nori: Mobile and embedded databases. In Proc. of ACM International Conference on Management of Data, 2007.

[12] P. Pucheral, L. Bouganim, P. Valduriez, C. Bobineau: PicoDBMS: Scaling down Database Techniques for the Smart card. *Very Large Data Bases Journal*, 10(2-3), 2001.

[13] Renesas Inc.: X Mobile Card, 2007. http://america.renesas.com/media/products/security/x-mobilecard/literature/X-MobileCardProductBrief.pdf

[14] The SmartRight Content Protection System. http://www.smartright.org. 2007.

[15] Trusted Computing Group. http://www.trustedcomputing.org. 2007.

[16] H. Yu, D. Agrawal, A. El Abbadi: Tabular Placement of Relational Data on MEMS-based Storage Devices. In Proc. of International Conference on Very Large Databases, 2003.