A Secure Search Engine for the Personal Cloud

Saliha Lallali [1,2] Nicolas Anciaux [1,2] Iulian Sandu Popa[1,2] Philippe Pucheral[1,2]
[1]INRIA Rocquencourt
78153 Le Chenay Cedex, France
FisrtName.LastName@inria.fr
[2]Université de Versailles Saint-Quentin en Yvelines
78035 Versailles Cedex, France
FirstName.LastName@uvsq.fr

ABSTRACT
The emerging Personal Could paradigm holds the promise of a Privacy-by-Design storage and computing platform where personal data remain under the individual's control while being shared by valuable applications. However, leaving the data management control to user's hands pushes the security issues to the user's platform. This demonstration presents a Secure Personal Cloud Platform relying on a query and access control engine embedded in a tamper resistant hardware device connected to the user's platform. The main difficulty lies in the design of an inverted document index and its related search and update algorithms capable of tackling the strong hardware constraints of these devices. We have implemented our engine on a real tamper resistant hardware device and present its capacity to regulate the access to a personal dataspace. The objective of this demonstration is to show (1) that secure hardware is a key enabler of the Personal Cloud paradigm and (2) that new embedded indexing and querying techniques can tackle the hardware constraints of tamper-resistant devices and provide scalable solutions for the Personal Cloud.

Categories and Subject Descriptors
E.2 [**Data Storage Representations**]: Composite structures; H.3.3 [**Information Search and Retrieval**]: Search process; H.3.1 [**Content Analysis and Indexing**]: Dictionaries, Indexing methods.

General Terms
Algorithms, Management, Performance, Design, Security.

Keywords
Embedded Search Engine, Access Control, Personal Cloud.

1. INTRODUCTION
We are witnessing an exponential increase in the acquisition of personal data by administrations and companies, and by the individuals themselves e.g., through quantified-self devices [1, 10]. In fine, all this data ends up in servers where it remains organized in *silos*, each under the control of their data collector. However, the pressure increases to link data between different silos thereby increasing business opportunities. Hence the strategy of major internet companies that search to horizontally control multiple data silos (domains of interest, leisure, geolocation, health, etc.). However, the PRISM affair[1] has made clear that centralizing and processing all one's data in a single server (or multiple servers controlled by a single actor) introduces a major threat on privacy. Moreover, this introduces a monopolistic situation impeding the free development of many valuable applications by others.

To face this situation, the World Economic Forum formulates the need for *a data platform that allows individuals to manage under their control the collection, usage and sharing of their data*[2]. This is precisely what the Personal Cloud paradigm aims to provide. Because it gives a tangible source of trust, the *home cloud* is the most emblematic form of Personal Cloud. It can be thought of as a dedicated box connected to the user's internet gateway, equipped with storage, computing and communication facilities [17], running a personal server and acquiring personal data from multiple sources [3, 22]. This personal server is in charge of organizing the personal dataspace in a document database style to ease its management, to allow crossing data from multiple "local silos" and to protect it against loss, theft and abusive use. Many projects and startups currently investigate this direction, e.g., OpenPDS[3], CozyCloud[4], Younity[5], Lima[6], OwnCloud[7], Tonido[8].

Leaving the data management control into the user's hands pushes the security issues to the user's computing platform as well. Documents can still be stored encrypted in Cloud servers but these servers never get access to the cryptographic keys and their role shrinks to guaranteeing the availability and resiliency of encrypted data. Hence, encryption/decryption, metadata management (e.g., indexing and search) and access control management is under the responsibility of the Personal server. This charge is paramount considering that all silos of personal data are now grouped in the user's hands.

This demonstration precisely tackles this challenge. The proposed solution combines a traditional home cloud data system organizing the document dataspace and a secure hardware-based co-server managing the encryption/decryption of documents and enforcing the access control rules. Our objective is to demonstrate that (1) such combination of software and tamper-resistant hardware can provide the individuals with the tangible source of trust required for a wide Personal Cloud adoption and (2) new search engine techniques embedded in tamper-resistant hardware can be devised to manage large-scale personal dataspaces with surprisingly good performance.

SIGMOD'15, May 31 - June 04, 2015, Melbourne, VIC, Australia

http://dx.doi.org/10.1145/2723372.2735376

[1] See e.g. the Wikipedia page on PRISM surveillance program

[2] The World Economic Forum. Rethinking Personal Data: Strengthening Trust. May 2012.

[3] OpenPDS: http://openpds.media.mit.edu/

[4] CozyCloud: https://www.cozycloud.cc

[5] Younity: http://getyounity.com/

[6] Lima : https://meetlima.com/

[7] OwnCloud : https://owncloud.org/

[8] Tonido : http://www.tonido.com/

2. ARCHITECTURE AND SCENARIOS

Let us consider the Secure Personal Cloud Platform of Alice as pictured in Figure 1. The main component is a home cloud data system gathering personal data from multiple sources (employer, banks, hospitals, commercial web sites, etc) and devices (smart meters, quantified-self devices, smartphones, cameras, etc.). The Personal Cloud can be implemented by any type of computing platform with storage facilities such as a set-up box or a plug computer. This data system is associated to a secure co-server which can be hosted by any type of tamper-resistant devices flourishing today, e.g., Mobile Security Card[9] (produced by Giesecke & Devrient), Personal Portable Security Device[10] (produced by Gemalto and Lexar), Multimedia SIM card or Secure Portable Token [4, 18]. Whatever its commercial name and form factor, a tamper-resistant device, called *secure token* hereafter, embeds a secure microcontroller (e.g., a smart card chip) connected to a large NAND Flash memory (e.g., a SD card) and can communicate with a host through a USB, Bluetooth or Ethernet port.

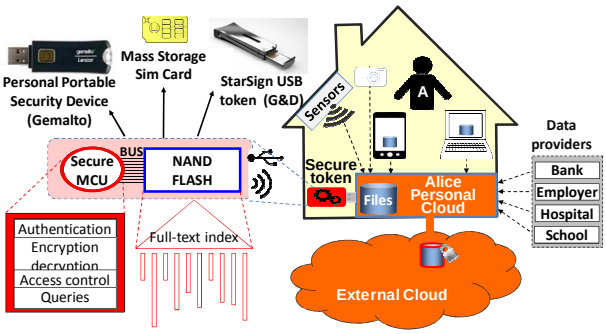

Figure 1. Secure Personal Cloud platform.

The documents entering Alice's dataspace can be any form of files (pictures, text files, pdf files, mails, data streams produced by sensors, etc). To help Alice query this diversity of documents, we consider the use of a simple keyword search engine, in the spirit of Google desktop or Spotlight. Terms are extracted from the file content and from metadata describing it (e.g., name, type, date, creator, tags set by the user herself). Since keywords seem the most convenient way to query Alice's dataspace, it make sense to let her express access control rules with a similar paradigm, that is through logical expressions on terms associated to documents. Hence, a query issued by a user or an application can be any form of term search expression, with a ranking function (e.g., *tf-idf* [23]) identifying the top-k most relevant documents (see Section 3). Only the documents granted to the user/application must appear in the query result (e.g., a query issued by the *mailer* application can only see documents tagged with the terms *email* or *contact*).

The question is how to make this architecture secure? To answer information retrieval queries, search engines usually rely on an inverted index containing all relevant terms extracted from the content and metadata of all the documents in the collection. This inverted index is thus as sensitive as the document collection itself and must be protected accordingly. However, Alice (as most of the typical users) is not a world-class security expert. Therefore, her home computing platform cannot reasonably be considered trusted. The unique source of security is provided by the secure token connected to her home platform (see Figure 1). This means that the inverted index and the query and access control engine must be embedded in the secure token to avoid any data disclosure at query execution time, except the final result. The high level of security (and hence trust) of this architecture is rooted in the combination of hardware and software security and data decentralization, which we detail in Section 5.3.

Designing such embedded query and access control engine is however very challenging due to a combination of severe and conflicting hardware constraints from the secure tokens. To the best of our knowledge, no state of the art work can meet document insertion performance and query scalability at the same time in this context (see Section 3). Moreover none of them support file deletions and updates, nor consider access control, which are mandatory in the Personal Cloud context.

3. TECHNICAL CHALLENGES

Search Engine Requirements. We consider that the search engine of interest in our context has a similar functionality as a Google desktop for secure tokens. Hence, we use the terminology introduced in the Information Retrieval literature for full-text search. Then, a document refers to any form of data files, terms refers to any form of data/metadata elements, term frequencies refer to data/metadata element weights and a query is equivalent to a full-text search. Full-text search has been widely studied by the information retrieval community since decades (see [23] for a recent survey). The core problem is, given a collection of documents and a user query expressed as a set of terms $\{t_i\}$, to retrieve the k most relevant documents that the querier is granted access to. In the wide majority of works, the *tf-idf* score is used to rank the query results: $tf\text{-}idf\ (d) = \sum_{t \in Q} \log(f_{d,t} + 1) \cdot \log(N/F_t)$, where $f_{d,t}$ is the frequency of term t in document d, N is the total number of indexed documents, and F_t is the number of documents that contain t. This formula is given for illustrative purpose, the weight between $f_{d,t}$ and N/F_t varying depending on the proposals. Classically, full-text search queries are evaluated efficiently using a B-tree-like inverted index storing for each term t the number F_t and the list of $(d, f_{d,t})$ pairs.

In our context, the same index will be used to rank the documents pertaining to the query result and to filter out those which do not match the access control rule. For example, Bob can search in Alice's dataspace for the k most relevant files containing (or tagged with) terms 'vacation', '2014', 'France' but will see only those which have been tagged by Alice with the term 'friends'.

Secure Tokens' Hardware Constraints. As mentioned earlier, the inverted index and the query and access control engine must be embedded in the secure token to enforce the security of the Personal Cloud. Whatever their form factor and usage, secure tokens share strong commonalities in terms of data management. Indeed, a large NAND Flash storage is used to persistently store data, metadata and indexes, and a microcontroller (MCU) executes the embedded code, both being connected by a bus. Hence, the architecture inherits hardware constraints from both the MCU and the Flash memory. Note that the tamper-resistance of the MCU provides a high level of protection against physical and side-channel attacks, and in particular for the data residing in

[9] http://www.gi-de.com/en/products_and_solutions/products/strong_authentication/Mobile-Security-Card-31488.jsp

[10] "Smart Guardian" http://cardps.com/product/gemalto-smart-guardian or "Smart Enterprise Guardian" http://cardps.com/product/gemalto-smart-enterprise-guardian

RAM since the RAM memory is located inside the MCU. Also, all the persistently stored data in the NAND Flash is cryptographically protected.

The MCU usually has a low power CPU, a tiny RAM (few KB), and a few MB of persistent memory (ROM, NOR or EEPROM) used to store the embedded code and secrets (e.g., cryptographic keys). The NAND Flash component (either raw NAND Flash chip or SD/micro-SD card) also exhibits strong limitations [7, 9]. In NAND Flash, the minimum unit for a read and a write operation is the page (usually 512 Bytes, also called a sector). Pages must be erased before being rewritten but the erase operation must be performed at a block granularity (e.g., 256 pages). Erases are then costly and a block wears out after about 10^4 repeated write/erase cycles. In addition, the pages have to be written sequentially in a block. Therefore, NAND Flash badly supports random writes.

State-of-the-Art solutions. Tiny RAM and NAND Flash persistent storage introduce conflicting constraints. Indeed, the index maintenance generates costly random writes in Flash which can be alleviated only at the expense of an extensive use of RAM. This leads to split state of the art on embedded search engines in two families. The *insert-optimized family* (e.g., [15, 16, 19, 21]) reaches insertion scalability thanks to a small indexed structure buffered in RAM (e.g., a small hash table) and sequentially flushed in Flash, thereby precluding costly random writes in Flash. This good insertion behavior is however obtained to the detriment of query scalability, the performance of searches being roughly linear with the index size in Flash (e.g., linear scan of very large hash buckets). Conversely, the *query-optimized family* (e.g., [2, 8, 12, 20]) reaches query scalability by adapting traditional indexing structures (e.g., B^+-Tree) to Flash storage, to the detriment of insertion scalability. The key idea in these approaches is to buffer the updates in log structures to amortize the Flash write cost in the same index nodes and to index this log in RAM to keep acceptable query performance. Hence, the number of random writes in Flash (linked to the log commit frequency) is roughly inversely proportional to the RAM capacity.

In conclusion, the existing solutions are scalable either for queries or insertions when considered in our context (please refer to [5] for an extensive evaluation of the representative full-text search methods). In addition, we are not aware of state of the art works addressing the crucial problems of random document deletions and of access control management in the context of an embedded search engine.

In the light of this discussion, the problem we address can be formulated as designing an embedded full-text search engine that meets three properties, namely:

(1) *Bounded RAM*: to consume a small and bounded amount of RAM to comply with the widest population of secure tokens (thereby precluding any solution where this consumption depends on the size of the document set).

(2) *Full scalability:* be scalable for queries and updates (insertion, deletion of documents) without distinction to comply with the widest usages.

(3) *Security:* to enforce access control rules and defeat tampering actions.

4. DESIGN OF THE SOLUTION

To tackle the challenge above, we proposed in [5] a novel indexing method that relies on the following design principles.

P1. Write-Once Partitioning: *Split the inverted index structure I in successive partitions such that a partition is flushed only once in Flash and is never updated.*

By precluding random writes in Flash, Write-Once Partitioning aims at satisfying update scalability. Considering the Bounded RAM property, the consequence of this principle is to parse documents and maintain I in a streaming way. Conceptually, each partition can be seen as the result of indexing a window of the document input flow, the size of which is limited by a predefined RAM_Bound. Therefore, I is split in an infinite sequence of partitions $<I_0, I_1,\ldots, I_p>$, each partition I_i having the same internal structure as I. When the size of the current I_i partition stored in RAM reaches RAM_Bound, I_i is flushed in Flash and a new partition I_{i+1} is initialized in RAM for the next window.

A second consequence of this design principle is that document deletions have to be processed similar to document insertions since the partitions cannot be modified once they are written. This means adding compensating information in each partition that will be considered by the query process to produce correct results.

P2. Linear Pipelining: *Compute each query Q with respect to the Bounded RAM property in such a way that the execution cost of Q over $<I_0, I_1,\ldots, I_p>$ is in the same order of magnitude as the execution cost of Q over I.*

Linear Pipelining aims at satisfying query scalability under the Bounded RAM agreement. A unique structure I (e.g., a traditional B^+-Tree) is assumed to satisfy query scalability by nature and is considered as providing a lower bound in terms of query execution time. Hence, the objective of Linear pipelining is to keep the performance gap between Q over $<I_0, I_1,\ldots, I_p>$ and Q over I, both small and predictable. Computing Q as a set-oriented composition of a set of Q_i over I_i, (with i=0,...p) would unavoidably violate the Bounded RAM property as p increases, since it will require to store all Q_i's intermediate results in RAM. Hence the necessity to organize the processing in pipeline such that the RAM consumption remains independent of p, and therefore of the number of indexed documents. Also, the term *linear* pipelining conveys the idea that the query processing must preclude any iteration (i.e., repeated accesses) over the same data structure to reach the expected level of performance. This disqualifies brute-force pipeline solutions where the *tf-idf* scores of documents are computed one after the other, at the price of reading the same inverted lists as many times as the number of documents they contain.

However, Linear Pipelining alone cannot prevent the performance gap between Q over $<I_0, I_1,\ldots, I_p>$ and Q over I to increase with the increase of p as (*i*) multiple searches in several small indexes are more costly than a single search in a large index and (*ii*) the inverted lists in $<I_0, I_1,\ldots, I_p>$ are likely to occupy only fractions of Flash pages, multiplying the number of Flash I/Os to access the same amount of data. A third design principle is then required.

P3. Background Linear Merging: *To limit the total number of partitions, periodically merge partitions in a way compliant with the Bounded RAM agreement and without hurting update scalability.*

The objective of partition merging is therefore to obtain a lower number of larger partitions to avoid the drawbacks mentioned above. Partition merging must meet three requirements. First the merge must be performed in pipeline to match the Bounded RAM property. Second, since its cost can be significant (i.e.,

proportional to the total size of the merged partitions), the merge must be processed in background to avoid locking the index structure for unbounded periods of time. Since multi-threading is not supported by the targeted platforms, background processing can simply be understood as the capacity to interrupt and recover the merging process at any time. Third, update scalability requires that the total cost of a merge run be always smaller than the time to fill out the next bunch of partitions to be merged.

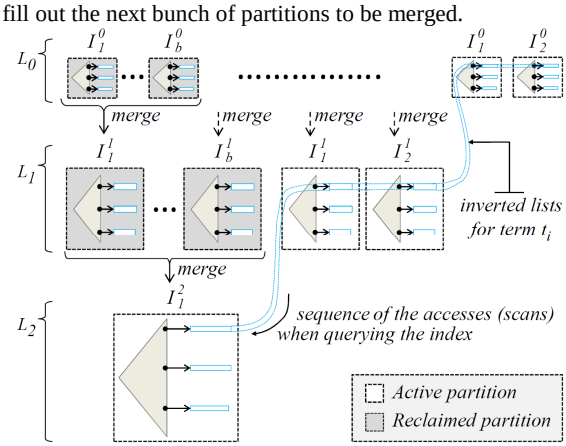

Figure 2. SSF index structure.

Taken together, principles P1 to P3 reconcile the Bounded RAM and Full scalability properties. These principles lead to the inverted index structure named SSF (Scalable and Sequential Flash index) pictured in Figure 2. This figure shows the incremental merging of partitions producing larger and larger indexes while generating only sequential writes in Flash. At each point of time, only the white partitions are active, the grey partitions being reclaimed by the merging process. For a same term *t*, the inverted list of (d, $f_{d,t}$) pairs traverses several partitions (from L_0 to L_2 in the figure), $I^0{}_2$ containing the most recent inserted documents and $I^2{}_1$ the oldest ones.

At query time, one buffer is allocated in RAM to sequentially scan the inverted list of each term involved in the query. Traditionally, a RAM variable is also allocated to each document *d* to compute its *tf-idf* score. This however hurts the bounded RAM property, the size of the document set being likely to be much larger than the available RAM. Conversely, we allocate a RAM variable only to the *k* documents having currently the best scores and compute the *tf-idf* scores in pure pipeline. Document ids reflecting the insertion ordering, the inverted lists are naturally sorted. Hence, the *tf-idf* computation sums up to a simple linear pipeline merging process of the inverted lists for all terms $t \in Q$ in each partition. The management of document deletions and updates and the fact that large documents can cross several partitions makes this process trickier. We refer the interested reader to [5] for a detailed presentation of the algorithms and performance evaluation.

The enforcement of access control rules is integrated seamlessly in the query computation. Access control rules are boolean expressions over terms present in or associated to the documents (e.g., authorize Bob to access all documents containing a given keyword or set of keywords or tags, as defined by tag-based access control models [6, 13, 14]). At query execution time, the inverted lists of terms involved in the access control rules are merged in pipeline with the inverted lists for all terms $t \in Q$. Then, a document enters in the set of the current *k* best documents if (1) its own *tf-idf* score is better than the worst of the *k* current best scores and (2) it satisfies the access control rule expression over its terms/tags.

5. DEMONSTRATION

In this section, we present our prototype platform and describe the demonstration scenario covering the security and the performance of the proposed solution for the Personal Cloud paradigm.

5.1 Demonstration platform

The Hardware Platform. The demonstration platform is an instance of the architecture presented in Figure 1. A PC plays the role of the home cloud data system organizing the document dataspace. In addition, the secure co-server manages the encryption/decryption of documents, indexes them, processes the queries and enforces the defined access control rules. The secure co-server runs on a tamper-resistant token produced by the ZED company and has an architecture representative for secure tokens (see Figure 3). Our secure token has the following characteristics: the MCU is equipped with a 32 bit RISC CPU clocked at 120 MHz, a crypto-coprocessor implementing AES and SHA in hardware, 128 KB of static RAM and 1 MB of NOR Flash. The MCU is connected to a smartcard chip hosting the cryptographic material and to a µSD card reader where the inverted index resides. The documents themselves are stored encrypted on the PC (i.e., the home cloud data system).

The secure token can communicate with the outside word either though the USB or the Bluetooth protocol. It is also equipped with a fingerprint reader providing a strong physical authentication of the queriers. The fingerprint reader can be used to identify the owner of the token that can either connect to her own token with all privileges or connect to another user's remote token with restricted privileges, which are regulated by access control rules established by the owner of the remote token.

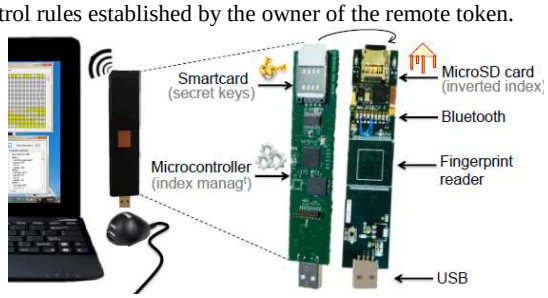

Figure 3. Secure token

The Graphical User Interface (GUI). The demonstration interface reflects the external APIs of our secure search engine enabling the insertion and deletion of sets of terms extracted from sets of documents, the keyword search and the definition of access control rules for the different subjects (see Figure 4, left part). When a given subject (e.g., an application, the data owner or a remote user) opens a session with the secure token, it first chooses an inverted index corresponding to a particular datasets. Then, it can add (or select) a set of files to (from) a (virtual) directory, to be inserted into (or deleted from) the index. Terms are extracted from these files and are completed with the appropriate tags before being inserted in the index embedded in the secure token. After insertion, the files appear encrypted on the host. The encryption key is automatically generated by the token from the document identifier and the secret key stored in the SIM smartcard and is unique for each file. Then, the encryption can be either performed by the host or by the token.

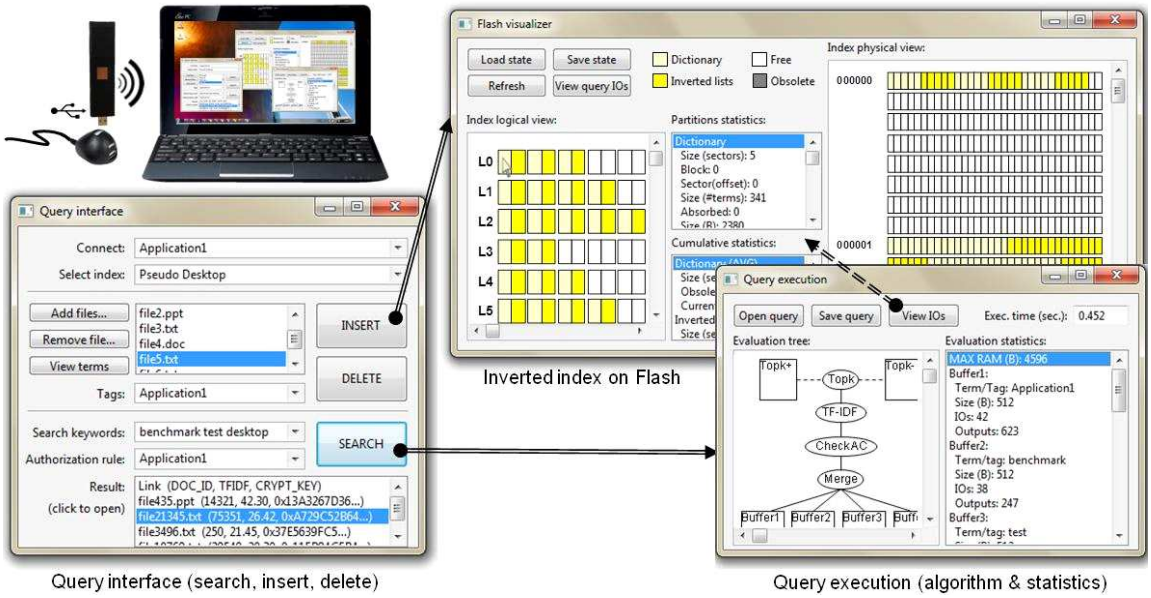

Figure 4. Demonstration graphical interface

The connected subject is able to issue keyword search queries to retrieve the relevant files. The relevant documents are ranked according to the *tf-idf* score and the top-*k* results are returned to the user. A link provides access to each result file, which can be decrypted on demand. The search queries conform to the access control rules. However, in the demonstration interface, a rule can be dynamically edited to show the effect on the result and the efficiency of the system whatever the access rule.

To demonstrate the behavior of the index, the interface also plots (see Figure 4, right upper part) the logical view of the index partitions and their corresponding physical view of the blocks/sectors in the Flash memory. Many useful statistics that describe the current state of each index partition, as well as the cumulative values maintained since the index creation, are also provided. For each insertion, deletion and search query, a synopsis of the underlying evaluation algorithm is also pictured (see Figure 4, right lower part) along with detailed statistics of their respective RAM consumption and IO patterns.

5.2 Demonstration scenario

The first part of the demonstration presents the motivation of the approach and shows the benefits of our search engine in terms of security and privacy protection in the context of the Personal Cloud. We use two representative collections of personal documents with their predefined query sets: a Pseudo-desktop Collection [11] composed of 27000 documents (emails, photos, pdf, docs, and ppts) and a large collection of emails (available at https://www.cs.cmu.edu/~enron/) containing more than 500 thousand emails to show the scalability of our platform. Representative applications (e.g., email) and users (data owner or remote users) are predefined with their respective access control rules (i.e., the files have been tagged accordingly). The attendees will be able to connect to the Personal Cloud as subjects granted with different access control rules, and will issue queries (insert, delete and search) to evaluate the usability and performance of the proposed technology. The enforcement of the access control rules will be illustrated with well prepared queries. Finally, the attendees may enroll their fingerprint on the token, insert a few files in the Personal Cloud tagged with their own keywords. We will then show that the files inserted by an attendee remain hidden to the other enrolled subjects.

The second part of the demonstration will focus on the search engine performance, scalability and compliance with the low RAM constraint. To demonstrate the scalability of the solution, we will use the large email data set. New files will be added or removed by the attendees and search queries will be executed on the indexed data set containing more than 500 thousand files. For each insertion, deletion and search query, the underlying execution plan is plotted with detailed statistics of memory consumption, IO costs and NAND Flash usage. The graphical user interface also shows the logical index structure and its physical representation in Flash memory. The IO pattern of each search query will be also visualized in the graphical interface, to better explain the scalability and efficiency of the search engine.

5.3 Demonstration results

Security. The primary expected result of this demonstration is to convince the audience that building a secure personal cloud platform where the evaluation of the access control rules is enforced by a secure token is within reach. The trustworthiness in the platform is rooted in two things. First, compared to a solution centralizing the personal data of millions of people in a single proprietary Cloud, the intrinsic decentralization of the Personal cloud decreases the incentive for attacks. Second, the use of secure hardware at the user's side strongly increases the difficulty and cost of an attack. Indeed, (1) the secure token provides a closed execution environment (no code can be installed and executed by the MCU but the original code and cryptographically signed versions of its upgrades), (2) the attacker must be in physical possession of the token to attack it, (3) tamper-resistance provides state of the art guarantees against physical and side-channel attacks, (4) the data at rest (stored in the NAND Flash) are cryptographically protected, (5) strong authentication is mandatory to unlock the personal server and no DBA or any other third party interfere in the loop. Hence, the ratio cost/benefit of an attack, which provides an accurate measure of the security level of a system, is greatly improved.

Bounded RAM. The second result is to demonstrate that a full-fledged search engine integrating access control can manage hundreds of thousands of documents while consuming kilobytes of RAM. The RAM consumption is indeed bounded by the number of terms involved in the query and in the access control rules and remains independent of the size of the document set. This results from the ability to execute any query in a pure

pipeline way even in the case where the document set is subject to updates (modifications and deletions of documents).

Full scalability. The third result is to show that improving query execution time is not always synonym of insertion/update degradation and vice-versa. This results from a sequential partitioned indexing structure providing linear performance for queries and insertions for a limited number of documents combined with a background reorganization process reaching logarithmic performance while the document set increases in size. The demonstration shows that the storage penalty incurred by this strategy is rather acceptable.

6. CONCLUSION

The emerging Personal Could paradigm holds the promise of a Privacy-by-Design storage and computing platform where personal data remains under the individual's control while being shared by valuable applications. However, this paradigm puts in the user's hand the responsibility of protecting all these data against multiple threats. While many personal cloud platforms are flourishing, riding the wave of repeated scandals blemishing the typical centralized management of personal data, none of them provide a tangible source of trust to the individuals. We hope that the platform demonstrated here contributes to the wider adoption of the Personal Could paradigm and emphasizes the interest of studying new database techniques embedded in secure hardware for the database community.

7. ACKNOWLEDGMENTS

This work is partially funded by project ANR-11-INSE-0005 KISS. Special thanks are due to Benjamin André from Cozy Cloud for fruitful discussions on the Personal Cloud paradigm.

8. REFERENCES

[1] Aggarwal, C. C., Ashish, N., and Sheth, A. The internet of things: A survey from the data-centric perspective. In Managing and mining sensor data, 2013.

[2] Agrawal, D., Ganesan, D., Sitaraman, R., Diao, Y. and Singh, S. Lazy-adaptive tree: An optimized index structure for flash devices. In *VLDB*, 2(1), 2009.

[3] Anciaux, N., Bonnet, P., Bouganim, L., Nguyen, B., Sandu Popa, I., and P. Pucheral. Trusted cells: A sea change for personal data services. In *CIDR*, 2013.

[4] Anciaux, N., Bouganim, L., Pucheral, P., Guo, Y., Folgoc, L. L., Yin, S. 2014. Milo-db: a personal, secure and portable database machine. In *Distributed and Parallel Databases*.

[5] Anciaux, N., Lallali, S., Sandu-Popa, I., Pucheral, P. A Scalable Search Engine for Mass Storage Smart Objects. *Technical report*, accessible at: http://www.prism.uvsq.fr/~isap/files/RT.pdf.

[6] Yeung, C., A., Kagal, L., Gibbin, N., Shadbolt, N. Providing Access Control to Online Photo Albums Based on Tags and Linked Data. AAAI Spring Symposium on Social Semantic Web: Where Web 2.0 Meets Web 3.0, 2009.

[7] Bjorling, M., Bonnet, P., Bouganim, L., and Jonsson, B. T. 2010. Uflip: Understanding the energy consumption of flash devices. In *IEEE Data Eng. Bull.*, 33(4).

[8] Debnath, B., Sengupta, S., and Li, J. Skimpystash: Ram space skimpy key-value store on flash-based storage. In *SIGMOD*, 2011.

[9] Diao, Y., Ganesan, D., Mathur, G., and Shenoy, P. J. Rethinking data management for storage-centric sensor networks. In *CIDR*, 2007.

[10] Huang, Y.-M., Lai, Y.-X. Distributed energy management system within residential sensor-based heterogeneous network structure. In *Wireless Sensor Networks and Ecological Monitoring*, 3, 2013.

[11] Kim, J. Y. and Croft, W. B. Retrieval Experiments using Pseudo-Desktop Collections. In *CIKM*, 2009.

[12] Li, Y., He, B., Yang, R. J., Luo, Q., and Yi, K. Tree Indexing on Solid State Drives. *VLDB*, 3(1-2), 2010

[13] Mazurek, M., L. A. Tag-Based, Logical Access-Control Framework for Personal File Sharing. 2014. PhD thesis.

[14] Klemperer, P., Liang, Y., Mazurek, M., Sleeper, M., Ur, B., Bauer, L., Cranor, F., L., Gupta, N., and Reiter, M. Tag, you can see it!: using tags for access control in photo sharing. In *SIGCHI*, 2012.

[15] Tan, C. C., Sheng, B., Wang, H., and Li, Q. Microsearch: When search engines meet small devices. *Pervasive Computing*, 2008.

[16] Tan, C. C., Sheng, B., Wang, H., Li, Q. 2010. Microsearch: A search engine for embedded devices used in pervasive computing. *ACM Trans. Embed. Comput. Syst.*, 9(4).

[17] To, Q.-C., Nguyen, B., and Pucheral, P. Privacy-Preserving Query Execution using a Decentralized Architecture and Tamper Resistant Hardware. In *EDBT*, 2014.

[18] Tsiftes, N. and Dunkels, A. A database in every sensor. In *ACM SenSys*, 2011.

[19] Wang, H., Tan, C. C., Li, Q. 2010. Snoogle: A search engine for pervasive environments. *IEEE Transactions on Parallel and Distributed Systems*, 21(8).

[20] Wu, C.-H., Kuo, T.-W., and Chang, L.-P. 2007. An efficient b-tree layer implementation for flash-memory storage systems. *ACM Trans. Embed. Comput. Syst.*, 6(3).

[21] Yan, T., Ganesan, D., and Manmatha, R. Distributed image search in camera sensor networks. In *ACM SenSys*, 2008.

[22] Yap, K.-K., Srinivasan, V., Motani, M. Max. 2008. Wide area human-centric search of the physical world. In *ACM Transactions on Sensor Networks*, 4(4).

[23] Zobel, J. and Moffat, A. 2006. Inverted files for text search engines. In *ACM Comput. Surv.*, 38(2).