Confidentiality and Tamper-Resistance of Embedded Databases

Yanli GUO

SMIS Project, INRIA - Paris Rocquencourt

INRIA Junior Seminar

SMIS project: Secured and Mobile Information System

- **Ubiquitous computing and ambient intelligence entail embedding data in increasingly light and specialized devices**
  - Such devices exhibit severe hardware constraints to match size, security, power consumption and also production costs requirements
  - They can highly benefit from embedded database functionalities to store the data, analyze it, query it and protect it
- **To make information more accessible and being acquired in transparent manner → ubiquitous computing and ambient intelligence involve new threats for data privacy**
- **Thanks to a high degree of decentralization and to the emergence of low cost tamper-resistant hardware →** ubiquitous computing contains the seeds for new ways of managing personal/sensitive data

SMIS project: Secured and Mobile Information System

- **Research themes**
  - Embedded Data Management (storage & index model)
  - Access and Usage Control Models(data sharing/retention condition controls)
  - **Tamper-resistant Data Management** **(Tamper resistance? →** being resistant to tampering by either the normal users of a product, package, or system or others with physical access to it)

Outline

- Personal Data Server Approach
- Embedded Database
- Crypto-Protection for Embedded Database

Outline

- **Personal Data Server Approach**
- Embedded Database
- Crypto-Protection for Embedded Database

How personal data is managed today (1)

Increasing amount of personal data automatically gathered on servers

- To meet the requirements of versatile applications (healthcare, e-administration, public transportation etc.)
- Pushed by government agencies and large companies
- No alternative today

Increasing amount of personal data delivered electronically to individuals

- Copies of invoices, salary forms, insurance policies, diplomas, etc
- Either stored on PC …
- … or resort to Storage provider servers for convenience, as they made data more durable and easily accessible through the Internet

➔ Personal data ends up in central servers anyway

How personal data is managed today (2)

Expected benefits

- **Data completeness, high availability, fault tolerance**
- Provide better services when the data is well organized, described, queryable

Does the benefits outweigh the privacy risk ?

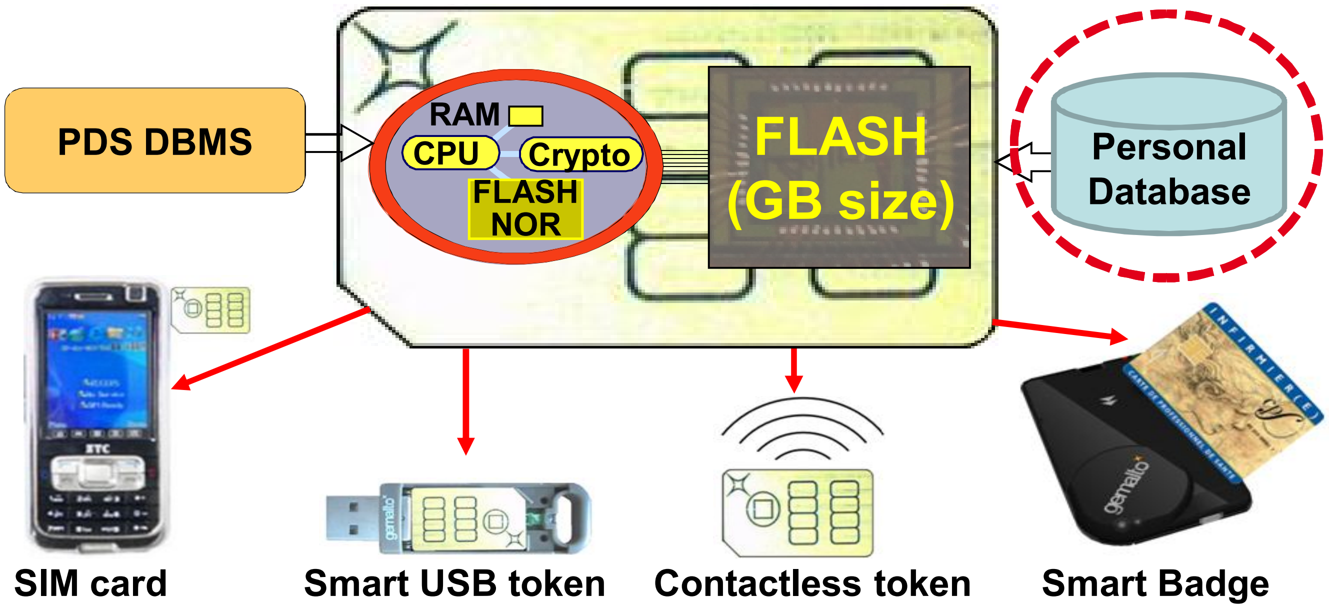
- Many examples of privacy violations due to **negligence** (DataLossDB), **abusive usage**, internal / external **attacks** (CSI/FBI)
- Data are out of the control of the data owners, while central servers do not provide ultimate privacy guarantee

Personal data server approach

Approach: Fully decentralized, where personal data is managed by a *Personal Data Server* (PDS)

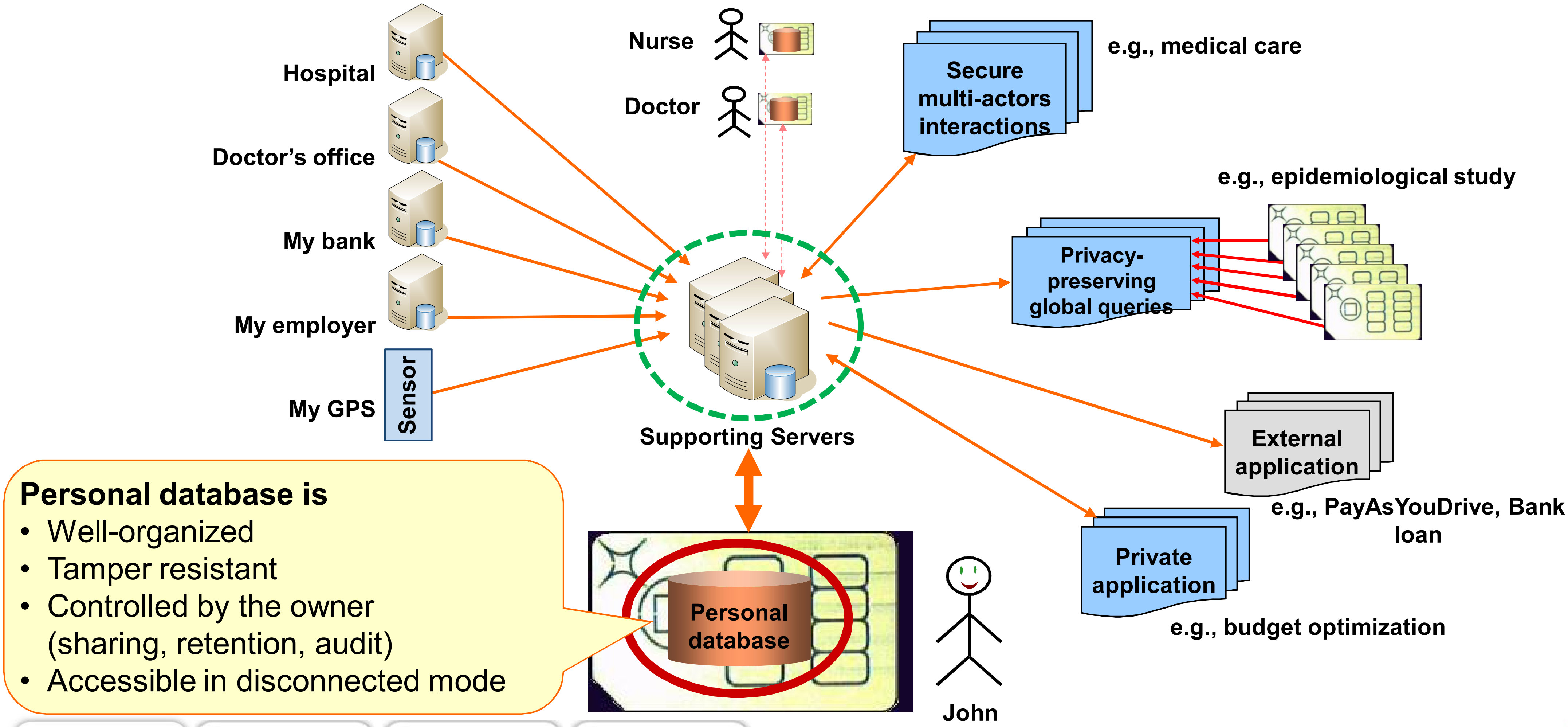
Secure Portable Token: (1) various form factors, (2) combines the tamper-resistant microcontroller which provides **secure processing** and **Gigabytes-sized NAND Flash storage**

How: Embed in the Secure Portable Token, software components capable of acquiring, storing and managing personal data



Personal data server vision

- **Objectives:** build a consistent information system based on a very large number of distributed and autonomous secure personal data servers, each hosting a **personal database**



Outline

- Personal Data Server Approach
- Embedded Database
- Crypto-Protection for Embedded Database

Design Requirements (1)

Microcontroller Constraints

- Small ratio between **RAM** size (32-128KB) / **NAND Flash** size (GBs)
- RAM remains scarce in the foreseeable future due to its poor density
  - RAM compete with other resources (e.g., CPU) in the same silicon die
- NAND storage (storing data) increases regularly
- Thus, the ratio RAM/NAND continues to decrease

➔ Require a massive indexing scheme for computing queries on GBs of data

Design Requirements (2)

NAND Flash Constraints (off-chip)

- R/W asymmetry
- Pages must be written sequentially within a Block
- Erase-before-rewrite (1 block vs. 1 page)
- Limited erase cycles

Random writes are difficult to manage

Generally handled by Flash Translation Layer → overheads/unpredictability

→ Design a database engine matching natively the NAND Flash constraints, e.g., proscribing random writes

Database Serialization Principle

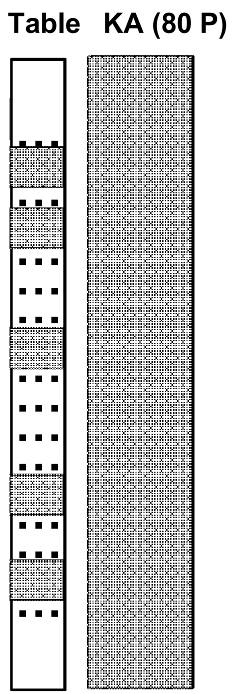
Objective: to break the implication between massive indexing and fine granularity random write patterns (conflicting constraints)

- Whole database (e.g., base tables, indexes, buffers, logs etc) is organized in pure sequential way, through ***Sequential Written Structures*** (SWS)
- SWS definition:
  - Data is written sequentially inside SWS
  - Written data never updated nor moved
  - Inside SWS, allocated Flash blocks are fully reclaimed and no partial Garbage Collection

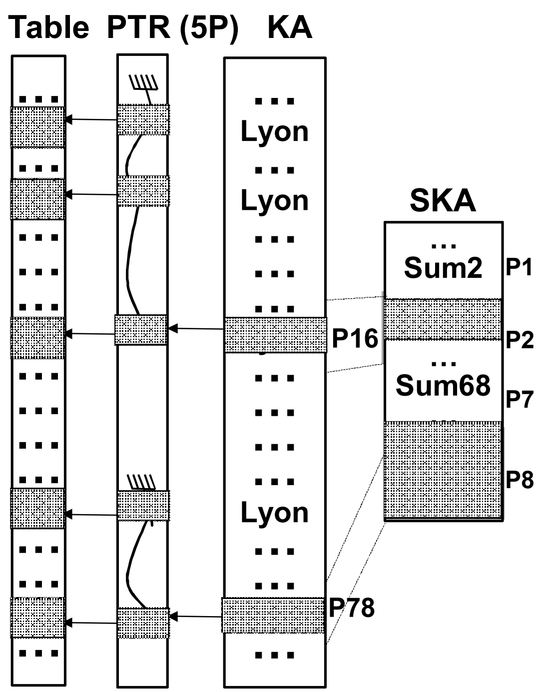
→ Serialization leads to satisfy Flash constraints by construction

- SWS proscribes random writes→ **overhead of Flash random writes avoid, Flash Translation Layer cost saved**
- Updates/deletes: logged in dedicated SWSs, handled during query processing (with specific algorithms)
- Selection indexes: as traditional indexes (e.g., tree-based or hash-based) lead to random writes, design specific cumulative indexes

Cumulative Indexes



Full Index
Scan (85 IOs)



Hybrid Skip
(15 IOs)

Database Stratification Principle

- **Objective**: cumulative indexes scales badly (being sequential)

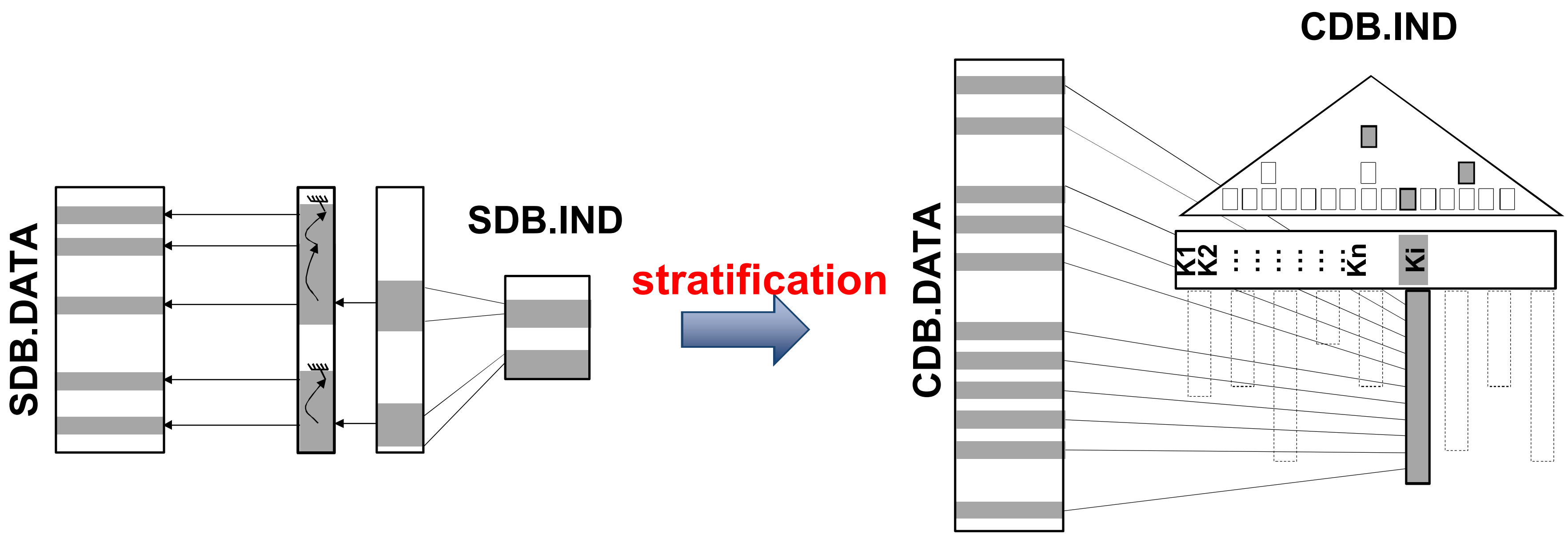
→ **Stratification to tackle the scalability issue**

- The idea is to transform a SWS database organization into a more efficient SWS database organization, i.e., performs as well as the state-of-the-art method ignoring Flash constraints
- Comply with the concept of SWS, i.e., proscribe random writes on Flash

How to stratify the database?

Stratification Process:

- Apply updates/deletes to the database and purge the log
- Reorganize the cumulative indexes into efficient clustered indexes (like B tree)
- Write the optimal reorganization of database into new stratum
- Reclaim the whole database before reorganization in the old stratum



Outline

- Personal Data Server Approach
- Embedded Database
- Crypto-Protection for Embedded Database

Threat model

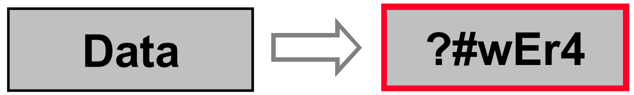
- **Microcontroller is tamper resistant**
- **NAND is vulnerable to:**
  - Snooping (e.g., deduce unauthorized information by examining the data) → **violate data confidentiality**
  - Tampering (e.g., modification, replaying, substitution)→ **violate data integrity**

→ Resort to crypto techniques to prevent any forms of attacks

Classical Crypto Countermeasures (1)

snooping:

- deduce unauthorized information → **use an opaque encryption scheme (e.g., CBC-AES)**



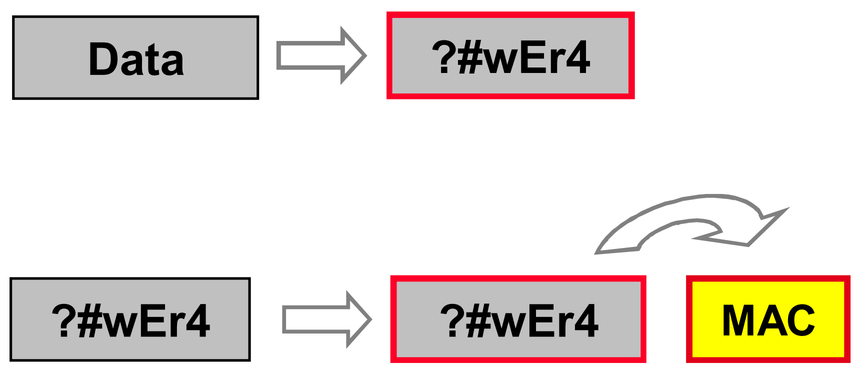
Classical Crypto Countermeasures (2)

snooping:

- deduce unauthorized information → **use an opaque encryption scheme (e.g., CBC-AES)**

modification:

- modifies (randomly) some data → **build and check a checksum (digest, e.g., MAC)**



Classical Crypto Countermeasures (3)

snooping:

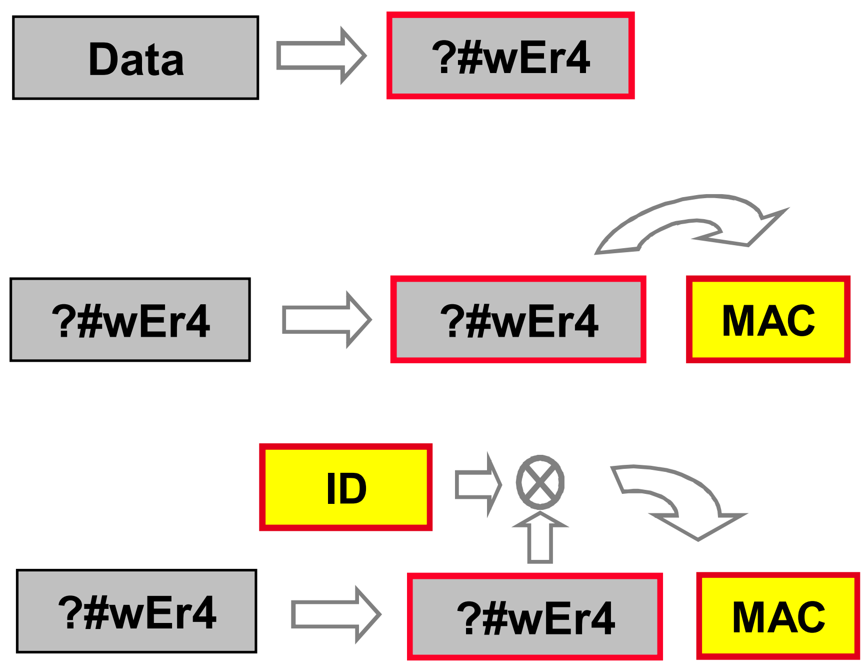
- deduce unauthorized information → **use an opaque encryption scheme (e.g., CBC-AES)**

modification:

- modifies (randomly) some data → **build and check a checksum (digest, e.g., MAC)**

substitution:

- replaces valid data with another valid data → **add and check an identifier (e.g., address)**



Classical Crypto Countermeasures (4)

snooping:

- deduce unauthorized information → **use an opaque encryption scheme (e.g., CBC-AES)**

modification:

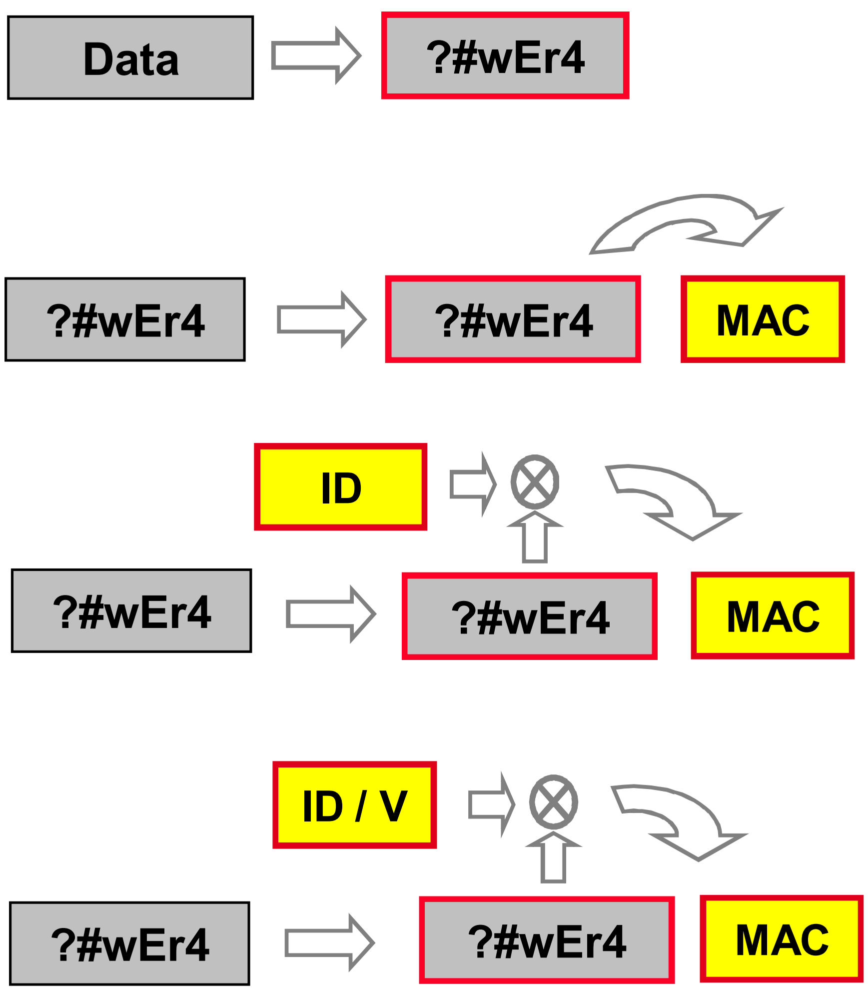
- modifies (randomly) some data → **build and check a checksum (digest, e.g., MAC)**

substitution:

- replaces valid data with another valid data → **add and check an identifier (e.g., address)**

replaying:

- replaces valid data with its older version → **add and check a timestamp (i.e., version number)**



Problem Statement (1)

Version management ?

- Need to store version numbers (for checking) to resist replay attack
- Few secure memory (e.g., 1 MB) is available for storing version ➔ **Keep minimal number of versions**
- Thus, let large number of data items share with the same version ➔ **high updating cost (when a single item updates)**

➔ Trivially answered thanks to the serialization/stratification

- All data in the same stratum has the same version
- Different versions of data item located in different stratums

➔use stratum number as version

Problem Statement (2)

Sequential search in embedded database

- Being prevailing in SWS based database (e.g., searching a key in a serialized index, i.e., whole page should be scanned)

Crypto performance

- Crypto operations are costly
- Typical values on secure chips:

	Read a page	Write a page
No Crypto	76 μs	301 μs
Enc only	460 μs (x6)	685 μs (x2)
Enc + MAC	846 μs (x11)	1071 μs (x3.5)

→ Design efficient crypto solutions for sequential search

Problem Statement (3)

Granularity of data in embedded database

- Embedded database relies on massive indexing scheme → Generate many fine granularity data (e.g., pointers, Bloom Filter accessed at bit granularity)

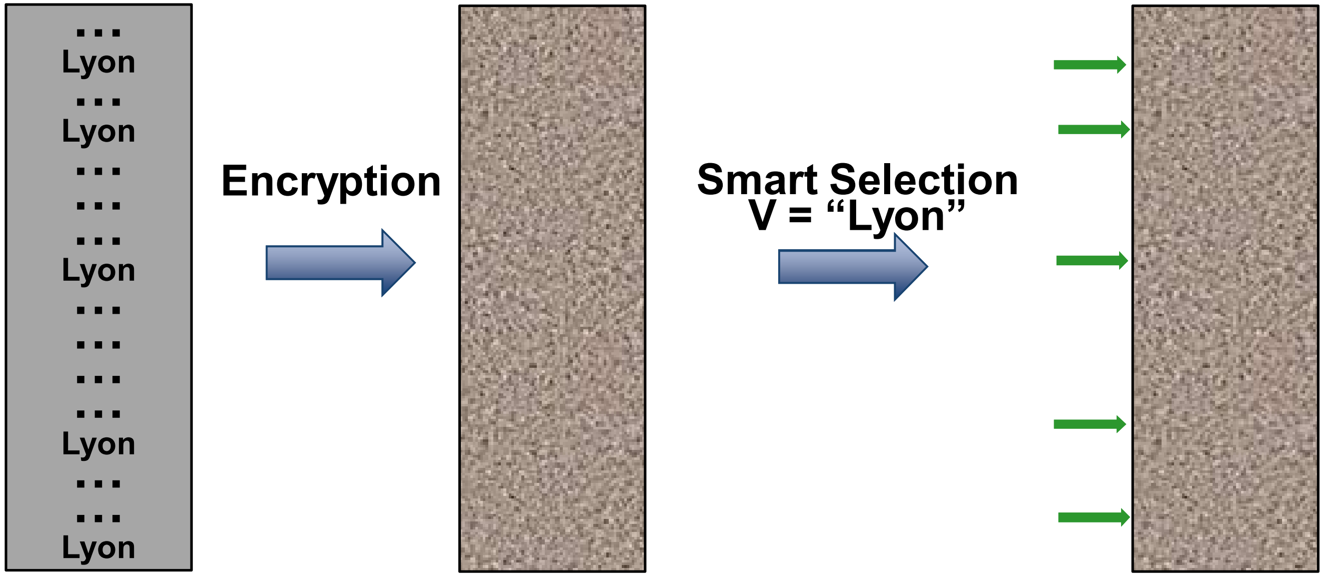
Granularity of cryptography primitives

- Encryption generally done on the block with 16 bytes
- MAC computation can be done
  - on 64 B blocks using cryptographic hash
  - on16 B blocks using encryption functions

→ Enforce confidentiality & integrity for fine granularity data efficiently

Smart Selection (1)

- **Objective:** search all occurrences of certain value within a set of values, without decrypting data, ensuring the completeness of the result



- Traditional encryption method require to **decrypt the whole set** before searching (with AREA)
- Smart Selection only needs **n+2** cryptographic operations for retrieving and ensuring the completeness of **n matching results**

→ **Do sequential search with minimal crypto overhead**

Smart Selection (2)

Example:

- **Find E(**0 || Lyon || NULL || 0) → @0
- **Find E(**1 || Lyon || @0 || 0) → @1
- **Find E(**2 || Lyon || @1 || 0) → NULL
- **Find E(**2 || Lyon || @1 || 1) → @2
- **Check MAC** for content of @2 → OK

…	MAC		E (0 || Lyon || NULL || 0)
E(0 || Lyon || NULL || 0)	MAC	Match	E (1|| Lyon || @0 || 0)
…	MAC		
E(1 || Lyon || @0 || 0)	MAC	Match	E (2 || Lyon || @1 || 0)
…	MAC		E (2 || Lyon || @1 || 1)
…	MAC		
…	MAC		
E(2 || Lyon || @1 || 1)	MAC	Match	Check MAC → END
…	MAC		
…	MAC		

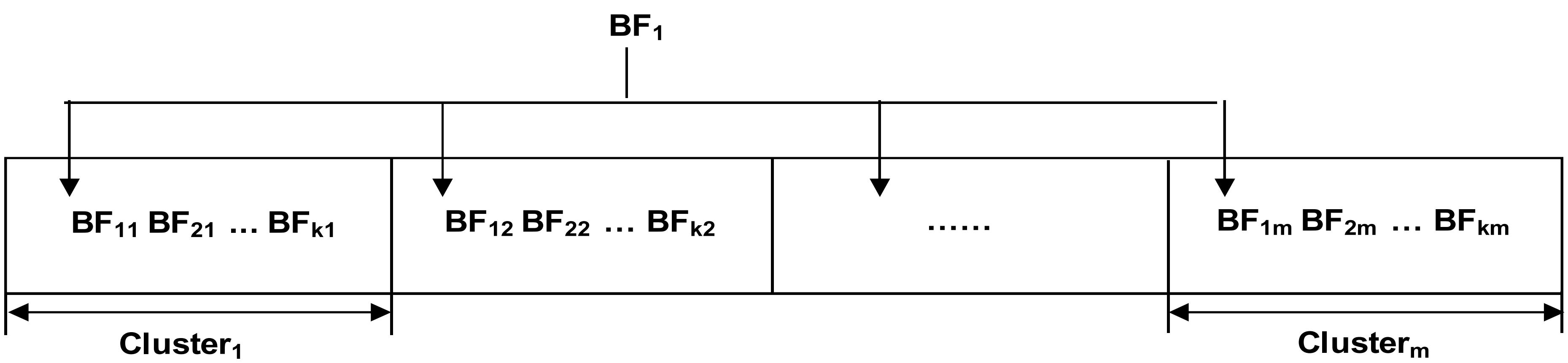
Questions ?

SMIS Project - team
INRIA Paris-Rocquencourt
http://www-smis.inria.fr/

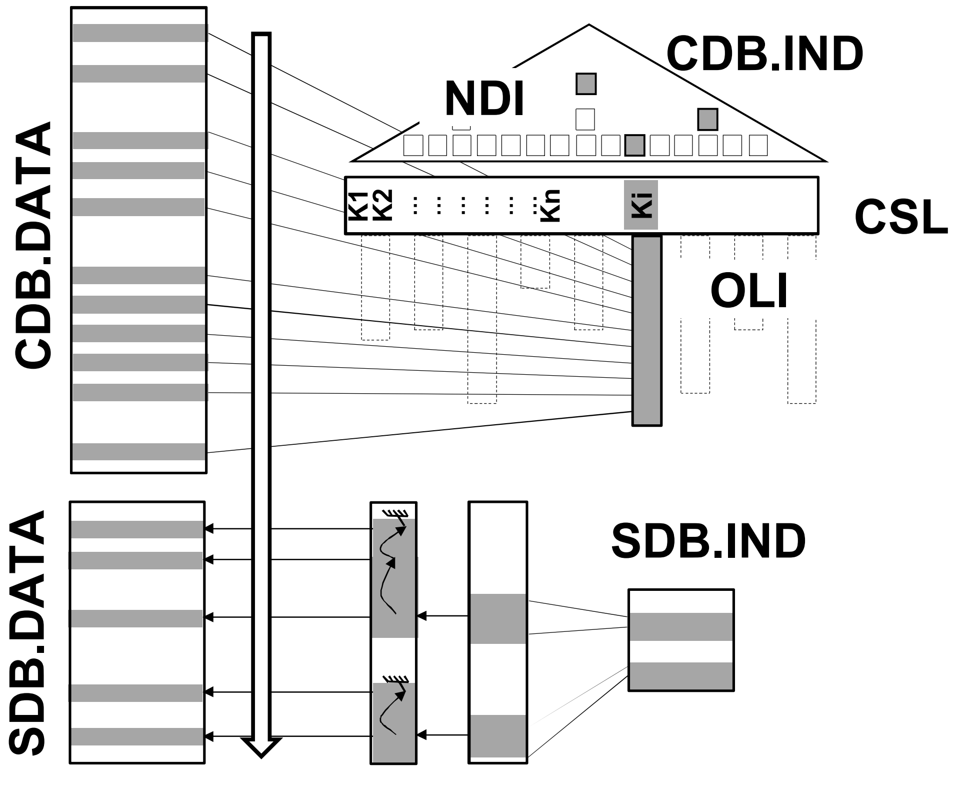
Thank You !

Data Placement

- Block cipher operates at a block granularity, it makes sense to **cluster data-of-interest within same blocks to reduce crypto overheads**
- **Requirements**: The data placement strategy should not violate the design rules of the PDS engine (i.e., disturb existing IO patterns or cause storage penalties)
- Example: SKA structure
  - BF is a bit array (e.g., *m* bits), and check few bits (e.g., 3 bits) in the array during testing
  - BFs inside the page are retrieved sequentially without skipping



Combining SDB and CDB



Combining SDB and CDB

CDB.IND	Clustered Index
NDI	Non Dense Index
CSL	Compact Sorted List
OLI	Ordered List of row Identifiers
CDB.DATA	Clustered Database
SDB.DATA	Serialized Database
SDB.IND	Cumulative Index

PlugDB Prototype (1)

PlugDB is an experimental project of secure and portable medical-social folder

- Objective: To improve the coordination of medical and social cares while giving the control back to the patient over how her data is accessed and shared
- Prototype: **simpler database organization**, i.e., no serialization, no stratification, basic crypto-protection done at sector granularity
- Being demonstrated at **SIGMOD 2010**

PlugDB Prototype (2)

Experimented in the field in the Yvelines District, France

USB 2 + Secure microcontroller
With a embedded web server and database management system