

UNIVERSITATEA „ALEXANDRU IOAN CUZA” IAȘI
FACULTATEA DE INFORMATICĂ



LUCRARE DE LICENȚĂ

Operații pe date encrypted la nivelul ORM-ului în contextul coregrafiilor encrypted

propusă de
Ioana-Maria Bogdan

Sesiunea: *Iulie, 2017*

Coordonator științific
Conf. dr. Lenuța Alboaie

UNIVERSITATEA ALEXANDRU IOAN CUZA IAȘI

FACULTATEA DE INFORMATICĂ

**Operații pe date encrypted la nivelul ORM-ului în
contextul coreografiilor encrypted**

Ioana-Maria Bogdan

Sesiunea: *Iulie, 2017*

Coordonator științific

Conf. dr. Lenuța Alboaie

DECLARAȚIE PRIVIND ORIGINALITATE ȘI RESPECTAREA DREPTURILOR DE AUTOR

Prin prezenta declar că Lucrarea de licență cu titlul „*Operații pe date encrypted la nivelul ORM-ului în contextul coregrațiilor encrypted*” este scrisă de mine și nu a mai fost prezentată niciodată la o altă facultate sau instituție de învățământ superior din țară sau străinătate. De asemenea, declar că toate sursele utilizate, inclusiv cele preluate de pe Internet, sunt indicate în lucrare, cu respectarea regulilor de evitare a plagiatului:

- toate fragmentele de text reproduse exact, chiar și în traducere proprie din altă limbă, sunt scrise între ghilimele și dețin referința precisă a sursei;
- reformularea în cuvinte proprii a textelor scrise de către alți autori deține referința precisă;
- codul sursă, imaginile etc. preluate din proiecte open-source sau alte surse sunt utilizate cu respectarea drepturilor de autor și dețin referințe precise;
- rezumarea ideilor altor autori precizează referința precisă la textul original.

Iași, 29.06.2017

Absolvent Ioana-Maria Bogdan

(semnătura în original)

DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul „*Operații pe date encrypted la nivelul ORM-ului în contextul coregrațiilor encrypted*”, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” Iași să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași, 29.06.2017

Absolvent Ioana-Maria Bogdan

(semnătura în original)

Abstract

În acest moment se estimează ca aproximativ 40 de înregistrări cu date private sunt copiate ilegal în fiecare secundă. Multe din aceste scurgeri de date se datorează faptului că informațiile sunt păstrate în text clar în bazele de date. O vulnerabilitate de securitate care oferă acces la bazele de date poate permite atacatorilor să extragă rapid cantități substanțiale de date.

Propunerea acestei lucrări este să fie modificat într-un mod cât mai puțin intruziv modul de exploatare a bazelor de date adăugând criptare pe datele sensibile, în așa fel încât să se diminueze riscurile ca datele private să fie copiate ilegal de administratorii bazelor de date sau de atacatorii ce exploatează breșele de securitate. Pentru a nu diminua eficiența bazelor de date, propunerea implică o soluție practică prin care o serie de interogări să se execute pe server direct pe date criptate, micșorând astfel suprafața de atac asupra acestor date sensibile. Față de alte abordări existente, contribuția descrisă constă în modificarea unui ORM (Object-relational mapping) și în modul nou de gestiune a parolelor de criptare.

Cuprins

1	Introducere	8
2	Imagine de ansamblu asupra datelor pe internet	10
2.1	Studii de caz	10
2.1.1	Datele personale distribuite pe internet	10
2.1.2	Scurgerea de date	11
2.2	Importanța datelor noastre și a bazelor de date	11
3	Aplicații similare și studii în domeniu	12
3.1	CryptDB	12
3.1.1	Schimbări arhitecturale necesare	12
3.1.2	Protocol	13
3.1.3	Vectori de atac	13
3.1.4	Concepte cheie	14
3.2	Google's Encrypted BigQuery Client	17
3.3	Microsoft Always Encrypted	17
3.4	Model teoretic	18
4	Tehnologii utilizate și concepte întâlnite	19
4.1	Baze de date relaționale (SQL)	19
4.2	Baze de date nerelaționale (NoSQL)	19
4.3	Redis	20
4.4	Node Package Manager (npm)	20
4.5	Object-relational mapping (ORM)	21
4.6	CRUD	21
4.7	Apersistence	22
4.8	Swarm Communication	23
4.8.1	Service-oriented architecture	23
4.8.2	Orchestrare	24
4.8.3	Coregrafie	24
4.8.4	Metafora <i>swarm</i>	25
4.8.5	Arhitectura <i>swarm</i>	25
4.8.6	Comunicarea într-un sistem de tip <i>swarm</i>	26

4.9	Private Data System	27
4.9.1	Imagine de ansamblu	27
5	Coregrafii criptate	28
5.1	Descrierea modelului propus	28
5.1.1	Operații asupra datelor encrypted la nivelul ORM-urilor	29
5.1.2	Interacțiunea dintre ORM și PDS	30
5.2	Avantajele modelului	35
5.3	Moduri de atac	35
6	Aplicație PoC folosind coregrafii encrypted	37
6.1	Use case	37
6.2	Arhitectura aplicației	38
6.2.1	Apresistence	38
6.2.2	PDS	39
6.3	Funcționalități oferite	39
6.3.1	Login & Register	39
6.3.2	Pagina personală	40
6.3.3	Echipa mea	41
7	Contribuții	42
7.1	Submisie ICCP	42
7.2	Contribuții <i>open source</i>	42
8	Concluzie	44
9	Bibliografie	45
	Appendices	48
Anexa A	Setarea mediului de lucru	48
A.1	Instalare <i>Node Package Manager</i>	48
A.2	Instalare <i>apersistence</i>	48
A.3	Instalare <i>redis</i>	48

1 Introducere

Termenul *Cloud Computing* este invocat din ce în ce mai des în ultimii ani în domeniul IT. *Cloud computing*-ul poate fi definit ca o paradigmă de programare la scară largă a unui mediu distribuit care se focusează pe economie de scară. Acest mediu distribuit este format dintr-o mulțime de resurse (cum ar fi capacitatea de stocare, platforme, servicii) abstracte, virtualizate, scalabile în mod dinamic, cu putere de calcul gestionabilă, livrabile la cerere clienților externi via Internet [1]. *Cloud computing*-ul este orientat spre servicii, nu spre aplicații, ceea ce nu numai că reduce cheltuielile de infrastructură și costul deținerii componentelor hardware, dar aduce și un plus de flexibilitate și performanță utilizatorilor.

Cloud computing-ul este foarte promițător pentru dezvoltatorii de aplicații. Cu toate acestea, există unele îngrijorări pentru utilizatori, fie ei persoane fizice sau întreprinderi cât și unele impedimente sau provocări pentru dezvoltatorii de cloud.

Una dintre provocările **dezvoltatorilor de cloud** este cea a comunicării cu mesaje asincrone. O soluție pentru comunicarea cu mesaje asincrone este descrisă în secțiunea 4.8.

Una dintre cele mai recurente îngrijorări ale **utilizatorilor de cloud** este cea a stocării într-un mod sigur a datelor.

Secțiunea 2 prezintă o imagine de ansamblu asupra datelor personale ale utilizatorilor în contextul *cloud computing*-ului și al comunicării datelor prin rețea. Cuprinde o serie de studii care arată aproximativ cantitatea datelor care sunt furnizate serviciilor, cât și o descriere despre cantitatea de date *leaked*.

Secțiunea 3 prezintă aplicații și studii similare sau care stau la baza cercetării descrise în această lucrare. Se prezintă modele de securizare a bazelor de date SQL și NoSQL, cât și un model teoretic care momentan nu este pus în practică.

Secțiunea 4 prezintă tehnologiile și conceptele întâlnite și însușite de-a lungul dezvoltării lucrării de față.

Secțiunea 5 descrie modelul cercetat și propus de această lucrare, iar secțiunea 6 descrie o aplicație *proof of concept* concepută folosind modelul coreografiilor encrypted descrise în secțiunile anterioare.

Secțiune 7 prezintă contribuțiile aduse odată cu dezvoltarea lucrării de față.

2 Imagine de ansamblu asupra datelor pe internet

În ziua de astăzi tehnologia este din ce în ce mai utilizată, astfel încât companiile sau corporațiile acumulează cantități enorme de date de la clienți, de multe ori aceste date fiind personale. Fiecare utilizator de internet și-a împărtășit datele personale cu cel puțin un provider de servicii (de exemplu, fiecare dintre noi are măcar un cont de email). Astfel, date precum nume, prenume, parola de la cont, eventual data nașterii sau numărul de telefon au ajuns în cel puțin o bază de date.

2.1 Studii de caz

2.1.1 Datele personale distribuite pe internet

Cei de la *globalwebindex* au publicat un studiu [2] care arată că fiecare utilizator de internet are în medie 5.4 conturi pe rețelele de socializare (de exemplu *Twitter*, *Facebook*, *Instagram*, etc.). Această medie diferă în funcție de vârstă, după cum se poate observa și în figura 1 extrasă din același articol menționat [2]. Pe rețelele de socializare de obicei furnizăm informații care pot fi folosite la *profiling*: preferințele noastre în materie de cărți, muzică, filme, etc. Acest studiu nu include și furnizorii de platforme care nu sunt de socializare, deci putem concluziona că acest număr de 5.4 va crește dacă luăm în considerare nu doar rețelele de socializare, dar și furnizorii de servicii (cum ar fi *Booking*, *eBay*, etc.). Un astfel de studiu mai general a fost făcut în Marea Britanie de cei de la *Experian* [3] care arată că oamenii între 25 și 34 de ani au în medie 40 de conturi online. În afară de date care ajută la *profiling*, din bazele de date ale marilor furnizori de servicii se pot obține până și informații clasificabile drept confidențiale (cum ar fi credit card number sau domiciliul, salariul).

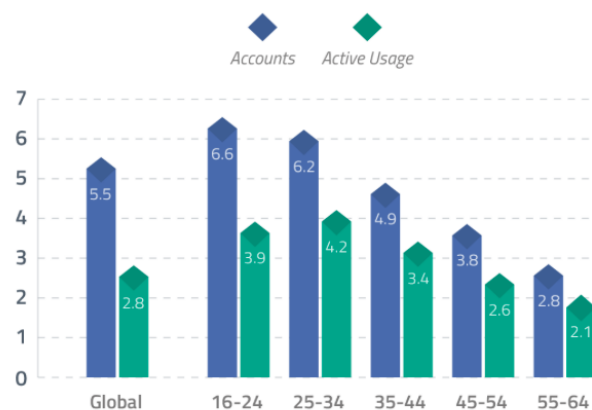


Figura 1: Numărul de conturi (total și active) pe rețelele de socializare în funcție de categoriile de vârstă.

2.1.2 Scurgerea de date

Cei de *breachlevelindex* [4] mențin statistici actualizate despre scurgerea de date. Din statisticile pe anul 2016 care pot fi găsite la [5] reiese faptul că au fost aproximativ 1800 de incidente raportate de scurgere de informație care au rezultat în aproximativ 1.4 miliarde de înregistrări scurse (aproximativ 44 pe secundă).

Pentru lucrarea de față, unul dintre cele mai relevante aspecte ale statisticii este cel din figura 2.



Figura 2: Procentul de date criptare (din procentul total de date scurse) care au fost scurse din bazele de date în anul 2016.

2.2 Importanța datelor noastre și a bazelor de date

Bazele de date sunt una dintre cele mai importante invenții preconizate de perioada anilor 1950 - 1970 (perioadă în care s-a făcut tranziția de la semnale analog la digital). Bazele de date facilitează o abstractizare a datelor, utilizatorului oferindu-i-se posibilitatea de a primi la cerere un subset relevant de date, fără a fi nevoie ca acesta să se preocupe de aranjarea datelor sau de menținerea persistenței acestora.

Studiile menționate mai sus arată că există o cantitate mare de date personale în una sau mai multe baze de date care nu sunt sub controlul nostru, ci a furnizorilor de servicii. Astfel de baze de date sunt o atracție foarte mare pentru atacatorii atât interni cât și externi.

Același lucru se întâmplă și în contextul *cloud computing*-lui, a cărui utilizare este în continuă creștere. Folosind servicii din cloud, furnizorii de servicii, cei la care ajung datele confidențiale ale clienților, s-ar putea folosi de *third party storage* (datele nu ar fi stocate intern de către furnizor, ci pe resurse din cloud). Aceste sisteme de gestiune a datelor la distanță au nevoie de anumite drepturi pentru administrare. Întrebarea care apare, în mod evident, este: *cum poate cineva să furnizeze datele clienților unei terțe părți, garantând totuși confidențialitatea datelor?*

3 Aplicații similare și studii în domeniu

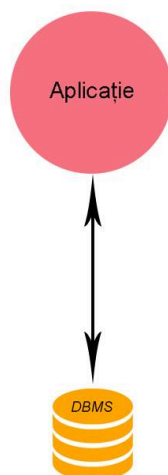
3.1 CryptDB

CryptDB este un sistem care asigură confidențialitate față de un atacator pentru aplicații care folosesc o bază de date SQL. *CryptDB* se bazează pe faptul ca serverul aplicației este diferit de cel al bazei de date (acesta fiind *design*-ul clasic al unei aplicații *database-backed*). În acest model, serverul aplicației ajunge să ruleze comenzi către DMBS (*Database Management System*) în numele utilizatorului. Ce se dorește prin *CryptDB* este ca aceste interogări efectuate asupra bazei de date să se execute asupra datelor criptate.

3.1.1 Schimbări arhitecturale necesare

CryptDB este proiectat să funcționeze ca un proxy între aplicație și baza de date. O aplicație poate fi un website, o aplicație care rulează pe dispozitive mobile sau o aplicație clasică de desktop. Cu alte cuvinte, o aplicație poate fi orice software care se conectează la o bază de date.

Design general



Cu Crypt DB

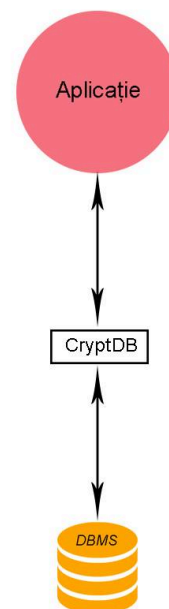


Figura 3: Schimbări arhitecturale necesare

3.1.2 Protocol

Protocolul *CryptDB* este format din 4 componente, după cum se poate vedea în *Figura 4*.

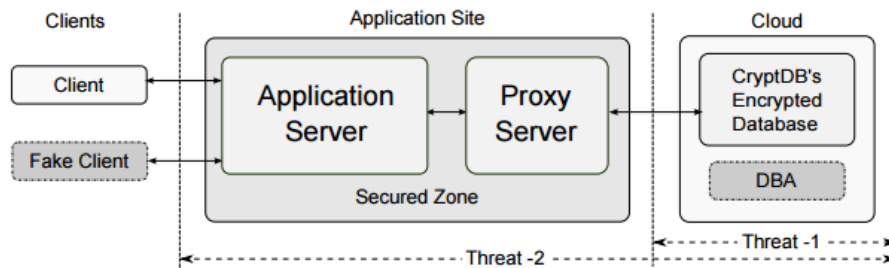


Figura 4: Protocolul *CryptDB* simplificat și vectorii de atac [6].

- **Clienții** sunt utilizatorii care furnizează datele către aplicație sau terțe părți cu acces autorizat la date. Clienții au nevoie ca procedurile standard ale bazei de date să proceseze datele într-un mod securizat. De asemenea, se vrea ca datele să fie disponibile numai entităților autorizate.
- **Serverul aplicației** este cel cu care interacționează utilizatorii. Acesta se conectează la baza de date protejată cu *CryptDB* prin intermediul serverului intermediar (*proxy*). Serverul aplicației preia datele de la utilizatori și le transmite mai departe *proxy*-ului.
- **Serverul *proxy*** este componenta nouă pe care o aduce protocolul *CryptDB*. *Proxy*-ul stochează cheia de criptare *Master Key* cu care, printre altele, ascunde numele coloanelor tabelurilor din baza de date. Serverul *proxy* este considerat entitatea de încredere a protocolului. Acesta deține pe lângă *MK (Master Key)* și o schemă adnotată a bazei de date în care sunt stocate nivelele de encripție a coloanelor. De asemenea, serverul *proxy* este responsabil de transformarea interogării într-un format în care aceasta să poată fi executată de către MySQL.
- **Baza de date *MySQL*** care se presupune a fi în cloud (serverul aplicației și baza de date nu se află pe aceeași mașină fizică). Cu ajutorul unor funcții definite de utilizator (*User Defined Functions* sau *UDF*) baza de date știe să execute interogările ascunse de către *proxy*. Serverul *MySQL* este considerat ca fiind parțial de încredere (se presupune că operațiile executate de acesta nu sunt alterate).

3.1.3 Vectori de atac

Vectorii de atac mitigați de *CryptDB* (după cum se pot vedea și în *Figura 4*) sunt:

1. datele utilizatorilor sunt ascunse față de un administrator de baza de date sau atacator care dorește să acceseze date confidențiale. Se presupune că atacatorul este pasiv, integritatea datelor păstrându-se.
2. un adversar are acces complet atât asupra serverului aplicației cât și asupra bazei de date (caz în care utilizatorilor care folosesc aplicația în perioada atacului nu li se poate garanta securitatea datelor).

3.1.4 Concepte cheie

Executarea interogărilor asupra datelor criptate presupune faptul că operațiile care vor fi efectuate asupra datelor sunt cunoscute *apriori*. Aceste operații sunt cele de bază oferite de SQL (verificarea egalității, inegalității, comparații, agregării, *join*-uri). Adăugările aduse sunt la nivelul serverului bazei de date (prin funcții definite de utilizator), nu la nivelul serverului aplicației, acesta rămânând nemodificat.

Această metodă mitighează primul vector de atac. Atât *proxy*-ul, cât și serverul aplicației sunt considerate ca fiind de încredere (*trusted*).

În funcție de tipul datelor pe care le stochează fiecare coloană, CryptDB oferă 6 modalități de criptare (Figura 5 [7] [8])

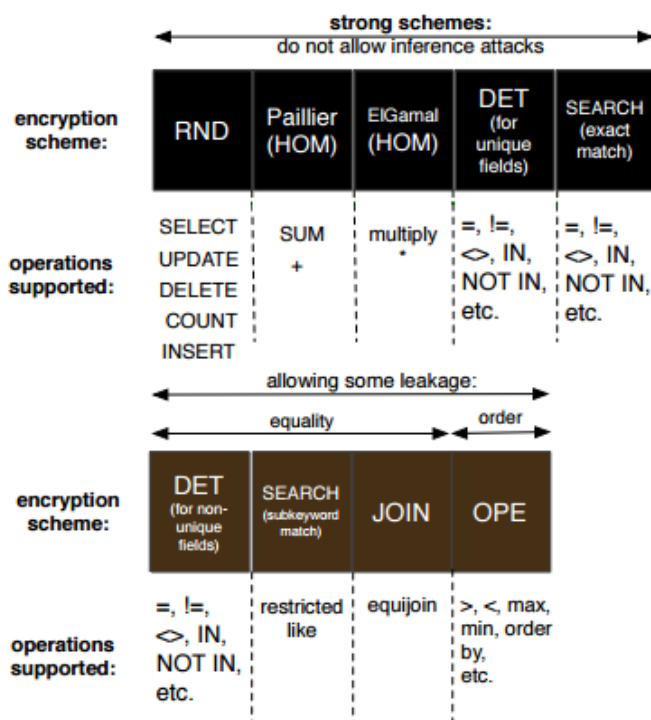


Figura 5: Protocoalele de criptare folosite de CryptDB și operațiile suportate de acestea

Random (RND) este cel mai puternic sistem de criptare folosit de CryptDB. La criptare se folosește un vector de inițializare (IV) generat *random*, ceea ce înseamnă că pentru același *plain-text* criptat în iterații diferite, textul criptat nu va fi același (criptarea este probabilistică). Dezavantajul lui RND este că nu permite nicio operație care necesită procesare asupra datelor (din moment ce nici măcar egalitatea nu este păstrată). Operațiile suportate sunt cele de SELECT, UPDATE, DELETE, COUNT și INSERT. Se poate observa că niciuna din aceste operații nu necesită vreun fel de agregare sau comparație a datelor din linii multiple. La nivel de implementare se folosesc AES și Blowfish.

Deterministic (DET) este varianta deterministă a lui RND. La nivelul implementării, se omite folosirea vectorului de inițializare. Acest mod de criptare permite operații de egalitate și inegalitate (pe lângă cele suportate deja de RND). La nivel de implementare se folosesc AES și Blowfish.

Order-preserving encryption (OPE), după cum sugerează și numele, păstrează ordinea după criptare. Adică, dacă $a < b$, atunci $OPE(a) < OPE(b)$. OPE este un sistem de criptare mai slab decât DET, deoarece dezvăluie ordinea inițială a valorilor din coloană. OPE permite operațiile uzuale de comparație. Schema folosită este conform articolului [9].

Homomorphic encryption (HOM) (criptare homomorfică) este o formă de criptare care permite calculelor să fie efectuate asupra textelor criptate, generând un text criptat. Acest rezultat, după decriptare, va fi egal cu rezultatul operației efectuată pe textele în clar [10]. Adică, dacă $a + b = c$, atunci $HOM(a) + HOM(b) = HOM(c)$.

Join (JOIN și OPE-JOIN) este o schemă separată de criptare care permite join-urile pe egalitate sau pe ordine. Schema este prezentată în articolul [8].

Word search (SEARCH) este un sistem de implementat după cum este descris în articolul [11]. Operația de *search* poate fi efectuată doar pe cuvinte întregi și nu se poate căuta după expresii regulate de lungime variabilă. *Search* este aproape la fel de sigur ca RND.

Criptarea ajustabilă pe baza interogării se folosește de mai multe straturi de criptare. Straturile de criptare poartă sugestiv numele de *onions*, această denumire fiind o metaforă pentru modul în care CryptDB împachetează datele. Scopul este de a folosi cea mai sigură schemă de criptare care totuși să permită executarea operației dorite. De exemplu, dacă se vrea egalitate ar trebui criptat cu DET iar dacă trebuie să se execute un join, va fi criptat corespunzător cu JOIN. Pentru a facilita abordarea descrisă mai sus este nevoie de o strategie adaptivă de criptare. Ideea folosită în CryptDB este de a folosi mai multe straturi de criptare pentru a facilita anumite operații. Modul în care se criptează este conform figurii 6 [8].

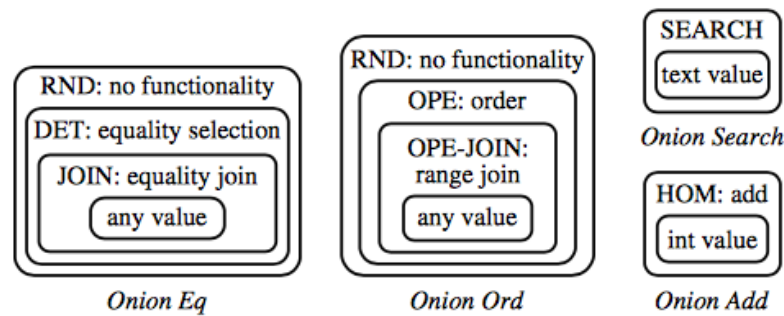


Figura 6: Modul în care CryptDB aplică straturile de criptare

Pentru fiecare layer de criptare se folosește aceeași cheie pentru intrările din aceeași coloană și chei diferite pentru coloane diferite.

Derivarea cheilor de criptare din parola utilizatorului este folosită pentru a asigura faptul că datele pot fi decriptate doar folosind un lanț de chei de decriptare care are ca prim element parola utilizatorului. Această tehnică garantează faptul că dacă un utilizator nu este activ în aplicație și atacatorul nu îi cunoaște parola, datele sale nu pot fi citite în text clar (chiar dacă ambele servere sunt compromise). Pentru a putea oferi acces la date între utilizatori se pot folosi niște adnotări speciale pentru a specifica *principalii* (utilizatori, grupuri) care au acces la datele respective, dacă datele sunt sensibile și cum se pot delega drepturi asupra datelor între *principalii*.

Fiecărui *principal* îi este asociată o cheie random. Dacă principalului B ii sunt delegate drepturi de ale lui A, cheia lui A este criptată folosind cheia lui B, ceea ce-i permite lui B să afle cheia de criptare a lui A și datele acestuia.

CryptDB criptează fiecare câmp adnotat ca având date confidențiale folosind *layere* de encripție (așa cum a fost descris mai sus). Cheile de criptare a straturilor sunt derivate din cheia unui principal, aceasta fiind formată dintr-o pereche de forma (*cheie publică*, *cheie privată*) și o cheie simetrică. De obicei se folosește cheia simetrică pentru a cripta datele și cheile altor principalii. Când un utilizator nu este *online* se folosește cheia publică, acesta urmând să decripteze cu cheia privată.

Când CryptDB criptează sau decriptează datele dintr-un rezultat al unei căutări, urmărește lanțul de chei care începe cu parola utilizatorului pentru a obține cheile dorite.

3.2 Google's Encrypted BigQuery Client

Cei de la Google dezvoltă un client pentru BigQuery [12]. Produsul este momentan experimental și oferă criptare pe partea de client a unui subset de tipuri interogări. Modelul lor este similar cu cel propus de CryptDB, folosind *onions of encryption*.

3.3 Microsoft Always Encrypted

Microsoft Always Encrypted [13] ajută la protejarea datelor sensibile din bazele de date Azure SQL și SQL Server. Datele sunt criptate la nivelul clientului și cheile de criptare nu sunt niciodată dezvăluite bazei de date sau serverului. Așadar, se creează o distincție între cei care dețin datele și ar trebui să le vadă, și cei care fac management al datelor și nu ar trebui să aibă acces la ele.

Configurarea se poate face la nivel de coloană, la fel ca în cazul CryptDB. Când se alege ca o coloană să fie criptată trebuie oferite informații despre algoritmul și cheile de criptare folosite pentru a proteja datele. Există două tipuri de chei: *column encryption key* care sunt folosite pentru criptarea datelor dintr-o coloană și *column master key* care criptează una sau mai multe chei de criptare.

În *figura 7* se poate observa cum s-a creat o cheie master (MyCMK - MyColumnMasterKey), o cheie de criptare pentru coloană (MyCEK - MyColumnEncryptionKey). La crearea cheii de criptare se poate observa că s-a menționat cheia master și algoritmul folosit pentru criptarea cheii MyCEK.

```
CREATE COLUMN MASTER KEY MyCMK
WITH (
    KEY_STORE_PROVIDER_NAME = 'MSSQL_CERTIFICATE_STORE',
    KEY_PATH = 'Current User/Personal/f2260f28d909d21c642a3d8e0b45a830e79a1420'
);

-----

CREATE COLUMN ENCRYPTION KEY MyCEK
WITH VALUES
(
    COLUMN_MASTER_KEY = MyCMK,
    ALGORITHM = 'RSA_OAEP',
    ENCRYPTED_VALUE = 0x01700000016C006F00630061006C006D0061006300680069006E0065002F00
);

-----

CREATE TABLE Customers (
    CustName nvarchar(60)
        COLLATE Latin1_General_BIN2 ENCRYPTED WITH (COLUMN_ENCRYPTION_KEY = MyCEK,
        ENCRYPTION_TYPE = RANDOMIZED,
        ALGORITHM = 'AEAD_AES_256_CBC_HMAC_SHA_256'),
    SSN varchar(11)
        COLLATE Latin1_General_BIN2 ENCRYPTED WITH (COLUMN_ENCRYPTION_KEY = MyCEK,
        ENCRYPTION_TYPE = DETERMINISTIC,
        ALGORITHM = 'AEAD_AES_256_CBC_HMAC_SHA_256'),
    Age int NULL
);
GO
```

Figura 7: Exemplu de creare a cheilor de criptare și a unui tabel [13]

La crearea tabelului *Customers* se poate observa cum unor coloane li se atașează o cheie, un tip de criptare (CustName cu criptare nedeterministă și SSN cu criptare deterministă), și un algoritm de criptare (AES pentru amândouă).

Always Encrypted suporta două tipuri de criptare: deterministă și nedeterministă.

3.4 Model teoretic

Când vine vorba despre baze de date criptare, există și modele teoretice care funcționează în cazul ideal. Un astfel de model este descris în articolul publicat de Craig Gentry, *Fully Homomorphic Encryption Using Ideal Lattices* [14]. Schema descrisă în articolul citat permite interogarea datelor fără a fi nevoie ca acestea să fie decriptate.

Dezavantajul pe care îl aduce o astfel de schemă vine din punct de vedere computațional. Unele operații durează mult, necesită multe resurse, ceea ce face modelul neaplicabil în practică momentan.

4 Tehnologii utilizate și concepte întâlnite

4.1 Baze de date relaționale (SQL)

O bază de date SQL reprezintă o modalitate de stocare a unor informații și date, în mod **structurat**, pe un suport extern (un dispozitiv de stocare), cu posibilitatea extinderii ușoare și a regăsirii rapide a acestora.

Acest model organizează datele conform modelului relațional propus de E. F. Codd în 1970 [15]. Datele sunt organizate în tabele (sau relații). Fiecare tabel este format din linii și coloane. Antetul tabelului reprezintă proprietățile entității pe care o descrie tabelul.

În *figura 8* se poate vedea un tabel dintr-o bază de date relațională care stochează informații despre angajați. Antetul reprezintă proprietățile (informațiile de interes) despre un angajat (nume, prenume, un identificator pentru poziția pe care o ocupă, salariul și numărul de telefon). Se poate observa, de asemenea, că fiecare angajat este identificat printr-un cod unic (*IdNum*) - cheie primară.

Employees Table					
IdNum	LName	FName	JobCode	Salary	Phone
1876	CHIN	JACK	TA1	42400	212/588-5634
1114	GREENWALD	JANICE	ME3	38000	212/588-1092
1556	PENNINGTON	MICHAEL	ME1	29860	718/383-5681
1354	PARKER	MARY	FA3	65800	914/455-2337
1130	WOOD	DEBORAH	PT2	36514	212/587-0013

Figura 8: Exemplu de tabel dintr-o bază de date relațională [16]

Limbajul folosit pentru a lucra cu astfel de baze de date poartă numele de *SQL (Structured Query Language)*

4.2 Baze de date nerelaționale (NoSQL)

O bază de date NoSQL reprezintă o modalitate de stocare a unor informații și date, în mod **nestructurat**, pe un suport extern (un dispozitiv de stocare), cu posibilitatea extinderii ușoare și a regăsirii rapide a acestora.

Așadar, o bază de date NoSQL este opusul unei baze de date relaționale, în acest model datele nemaivând structură cunoscută *apriori*.

Printre avantajele modelului nerelațional se numără [17]:

- **evitarea complexității nenecesare** impusă de bazele de date relaționale. În unele cazuri restricțiile sunt costisitoare.
- **scalabilitatea orizontală** este mai ușoară în cazul bazelor de date nerelaționale, cerință din ce în ce mai întâlnită în contextul *cloud computing* (fiind mai ușor de adăugat o mașină nouă decât de adăugat resurse suplimentare aceleași mașini)
- complexitatea și costul de a **gestiona clustere de baze de date** scad din moment ce, ne-maiavând o structură fixă, datele nu mai trebuie adăugate neapărat într-un anumit tabel. În cazul sistemelor mari de baze de date, tabelele sunt împărțite pe mai multe mașini care comunică între ele prin sisteme de tip *grid*.

Printre cele mai folosite baze de date nerelaționale se numără *MongoDB*, *Redis* sau *DynamoDB*.

4.3 Redis

Redis (**RE**mote **DI**ctionary **S**erver) [18] este o modalitate *open source* de a structura datele *in-memory* care poate fi folosită ca baza de date, cache și agent de mesaje. Suportă structuri de date precum șiruri de caractere, tabele hash, liste, seturi, bitmaps, *etc.*

Performanța ridicată pe care o are Redis se datorează faptului că lucrează cu date *in-memory*. Pentru a stoca datele persistent se poate scrie pe disk la intervale regulate de timp sau prin logarea tuturor comenzilor. Există și opțiunea de a renunța la persistență.

Popularitatea *Redis*-ului a crescut în ultimii ani, astfel încât distribuitorii de cloud au început să creeze servicii care se bazează pe Redis (cum ar fi Amazon ElastiCache [19]). Printre cei mai mari utilizatori de Redis se numără Twitter sau StackOverflow.

4.4 Node Package Manager (npm)

npm [20] este cel mai popular *package manager* pentru limbajul de programare *Javascript*. *npm* vine instalat odată cu *NodeJS* (runtime de Javascript) și este format dintr-o linie de comandă pentru clienți și o bază de date online (*npm repository*) unde sunt stocate toate pachetele publicate.

npm ajută programatorii de Javascript atât să distribuie codul creat de ei, cât și să refolosească în aplicațiile dezvoltate module publicate de alți membri ai comunității.

Pachetele distribuite de programatori prin *node package manager* rezolvă probleme punctuale, concrete. Printre cele mai populare module de *npm* se numără *express* cu ajutorul căruia

se pot crea ușor aplicații web, *Socket.IO* pentru comunicarea client-server print web sockets sau *mocha* - framework de testare.

4.5 Object-relational mapping (ORM)

Object-relational mapping (sau *ORM*) este un mecanism care permite adresarea, accesul și managementul obiectelor făcând abstracție de modul în care acestea relaționează cu datele sursă [21].

ORM-urile sunt folosite pentru a converti date între tipuri de sisteme incompatibile. Cel mai des sunt folosite în contextul programării orientate-obiect. Prin ORM-uri se creează impresia unei baze de date virtuale care poate fi folosită în limbajul de programare pentru care se optează.

Conceptual, ORM-urile translează reprezentarea logică a obiectelor într-o formă automatizată care poate fi stocată într-o bază de date, păstrând relația dintre acestea. Acest procedeu mai poartă și numele de *persistență*.

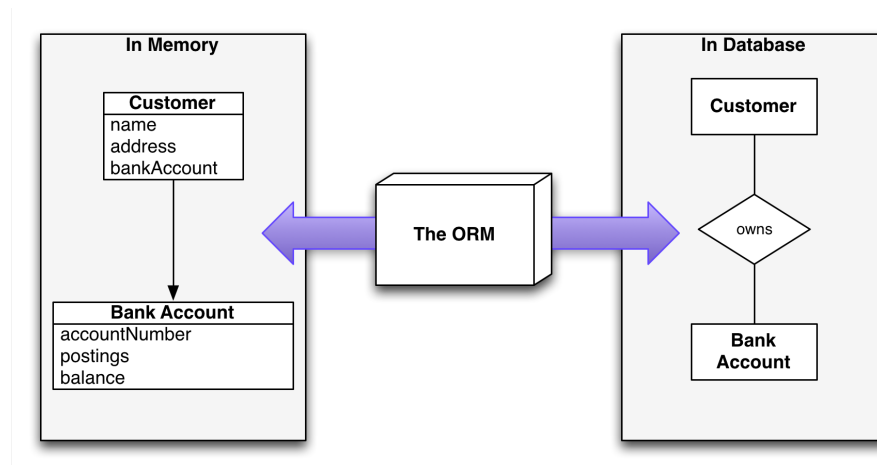


Figura 9: Exemplu de ORM [22]

4.6 CRUD

CRUD este un acronim de la **Create, Read, Update, Delete**. Acestea sunt cele 4 operații de bază pe care trebuie să le poată efectua orice capacitate de stocare persistentă (persistent storage).

Termenul de CRUD a început să fie folosit din de în ce mai mult și la nivelul interfeței aplicației. ORM-urile create peste baze de date implementează minimal aceste funcționalități.

- **Create:** adăugarea unei noi înregistrări într-o bază de date
- **Read:** citirea unor înregistrări din baza de date
- **Update:** modificarea unor valori ale unei înregistrări

- **Delete:** ștergerea unor înregistrări

4.7 Apersistence

Apersistence (abstract persistence) [23] este un modul de *npm* cu care se pot crea nivele de abstractizare între obiecte și posibile mixturi de baze de date relaționale și nerelaționale. Cu alte cuvinte, folosind *apersistence* se pot crea ORM-uri atât peste baze de date relaționale, cât și peste NoSQL.

Apersistence implementează ORM-uri peste Redis și MySQL.

Folosirea modului *apersistence* facilitează schimbarea bazei de date în funcție de nevoile aplicației. Astfel, o bază de date relațională și una nerelațională devin interschimbabile. Acest lucru ajută developerii de aplicații web să se concentreze mai mult pe funcționalitate decât pe modul de reprezentare a datelor. Câteva exemple de utilizare:

- **Declararea unei persistențe**

```
var redis = require("redis");
var redisConnection = redis.createClient()

var apersistence = require("apersistence");
var persistence = apersistence.createRedisPersistence(redisConnection);
```

Figura 10: Inițializarea unei persistențe peste Redis folosind modulul *apersistence*

```
var mysql = require('mysql');
var mysqlConnection = mysql.createConnection({
  host      : 'the_host',
  user      : 'username',
  password  : 'password',
  database  : 'database_name'
});

var apersistence = require("apersistence");
var persistence = apersistence.createMySqlPersistence(mysqlConnection);
```

Figura 11: Inițializarea unei persistențe peste MySQL folosind modulul *apersistence*

Figurile 10 și 11 arată modul de conectare la baza de date relațională MySQL și la cea nerelațională Redis.

Folosind funcțiile *createRedisPersistence* și *createMySqlPersistence* oferite de pachetul *apersistence* se declară persistențe peste bazele de date corespunzătoare. Ambele funcții primesc ca parametru un obiect conexiune la baza de date referită.

- **Înregistrarea unei persistențe**

În figura 12 se poate observa modul în care poate fi creat un ORM peste o bază de date. Asociind cu modelul relațional (figura 8), modelul creat cu ajutorul funcției *registerModel* poate fi văzut ca o reprezentare relațională. Cele două caracteristici din modelul minimal al unui angajat creat în figura 12 sunt *employeeNumber* drept cheie primară și *name*.

```
persistence.registerModel("EmployeeModel", {
  employeeNumber: {
    type: 'int',
    default: 1,
    pk: true
  },
  name: {
    type: 'string',
    default: 'John Doe'
  },
  getName: function(){
    console.log("Employee name: ", this.name);
  }
});
```

Figura 12: Model de persistență peste o bază de date

4.8 Swarm Communication

Comunicarea prin *swarm*-uri este o modalitate de a comunica în *cloud* care oferă scalabilitate dinamică [24]. Această modalitate constă într-un model de descompunere a serviciilor complexe în servicii mai mici.

Swarm se bazează pe o metaforă inspirată din natură și presupune că mesajele sunt entități *inteligente*.

4.8.1 Service-oriented architecture

SOA (*Service-oriented architecture*) este un tip de arhitectură software [25] care presupune distribuirea funcționalităților aplicației în unități distincte (servicii) care pot fi distribuite în rețea. Aceste servicii pot fi folosite împreună, funcționalitățile lor îmbinându-se pentru a crea un serviciu mai mare. Aceste servicii comunică în rețea prin protocoale de comunicare cunoscute și sunt independente de furnizor (*vendor*).

Așadar, un serviciu este o unitate de funcționalitate de sine stătătoare, care poate fi accesată la distanță (*remote*) și modificată independent.

SOA nu propune strategii de modularizare a aplicațiilor, ci sugerează moduri prin care microserviciile pot fi integrate pentru a dezvolta aplicații distribuite. Metodele de compoziție și

integrare a serviciilor folosesc, în mare, două stiluri arhitecturale: *orchestrarea* și *coregrafia*.

4.8.2 Orchestrare

Orchestrarea se folosește pentru a integra două sau mai multe servicii sau aplicații cu scopul automatizării unor procese sau sincronizarea datelor în timp real. [26].

Orchestrarea ajută la decuplarea codului, modulele nemaivând dependențe externe (care mereu s-au dovedit greu de întreținut). Astfel, se ușurează monitorizarea aplicației și mentenanța acesteia. Arhitectura bazată pe orchestrare propune sisteme **centralizate** în care fiecare serviciu este coordonat după reguli cunoscute *apriori*.

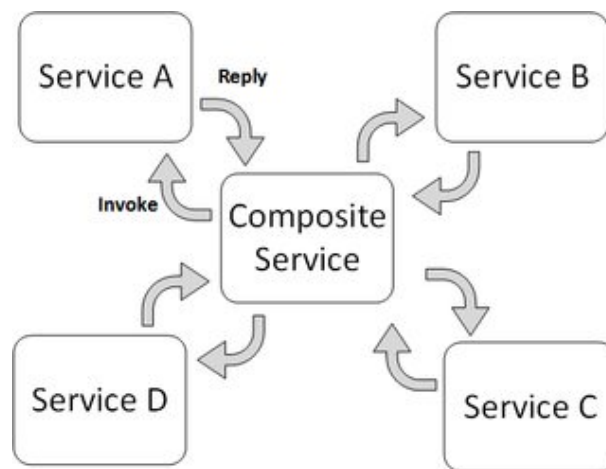


Figura 13: Orchestrare de servicii [27]

4.8.3 Coregrafie

În modelul coregrafiei se elimină coordonatorul din modelul orchestrării (rezultând un model **necentralizat**). Coregrafia se caracterizează prin o descriere globală a serviciilor participante, definită de schimbul de mesaje și reguli de interacțiune între două sau mai multe *endpoint*-uri.

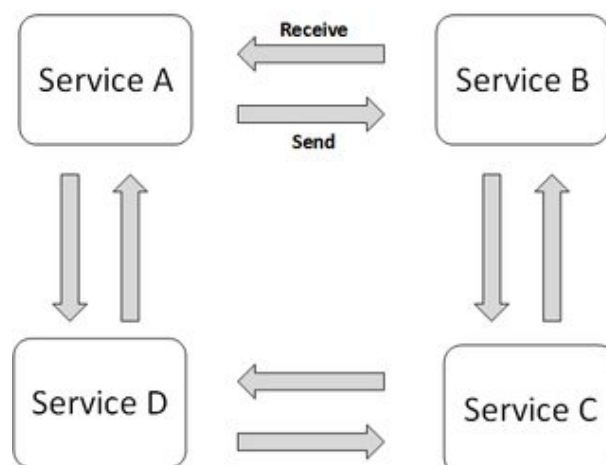


Figura 14: Coregrafie de servicii [27]

4.8.4 Metafora *swarm*

Atât numele (*swarm*), cât și modelul arhitectural sunt inspirate din natură. Un *swarm* este un mesaj care *roiește*, adică este transmis între noduri în rețea. Aceste noduri pot fi privite ca un stup. Albinele (cele care cară *mesajele*) se împrăștie prin natura cu diverse mesaje.

Transmiterea unor mesaje sau valori în rețea este comparată aici cu roitul albinelor care cară miere spre stup. După ce mesajul ajunge la destinație, sau albina la stup, se reia aceeași activitate.

4.8.5 Arhitectura *swarm*

Paradigma *swarm* este o arhitectură pentru microservicii bazată pe un șablon de comunicare numit la fel (*swarm*). În mod normal transferul de mesaje pleacă de la ideea că procesele (actori sau obiecte) dețin logica sistemului și primesc mesajele care sunt *dumb*. Comunicarea *swarm* schimbă perspectiva și gândește mesajele ca entități inteligente vizitând locuri *dumb*. Făcând această modificare s-a observat o calitate crescută a codului și un model diferit de descompunere a aplicațiilor în servicii mai mici. Folosind comunicarea prin *swarm*-uri se pot crea *ESB*-uri (Enterprise Service Bus) ușor scalabile. O astfel de implementare este SwarmESB [28]. Clientul poate să-și adauge propriile microservicii (numite *adaptori*) și să descrie cum acestea se îmbină pentru scenariul aplicației.

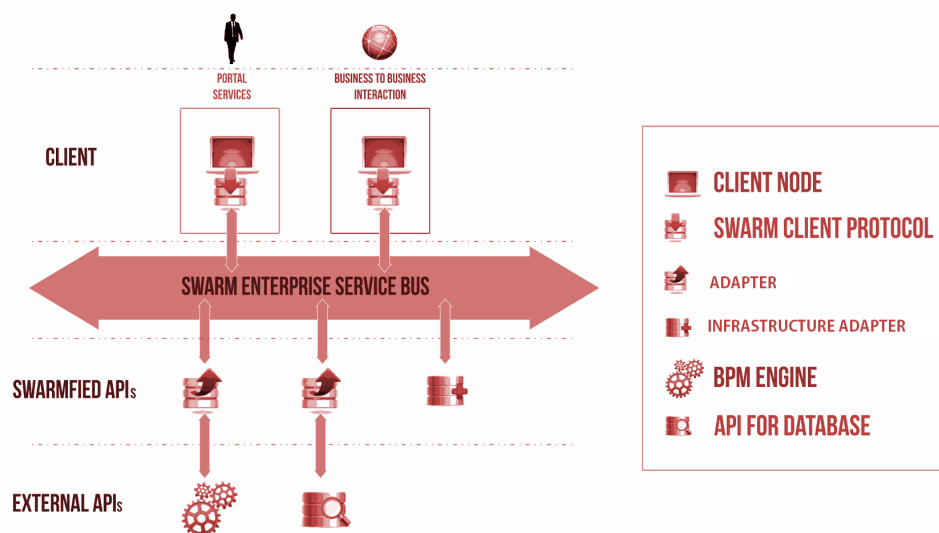


Figura 15: Arhitectura SwarmESB [28]

Scopul arhitecturii de tip *swarm* este același ca al șabloanelor de integrare oferite de SOA: compunerea de servicii cu comportamente diferite. *Swarm*-urile se îndreaptă în direcția *coregrafiei executabile* pentru a oferi din start o arhitectură facilă pentru securitate. Se aduce un element în plus modelului coregrafiei: nodurile sistemului distribuit acționează după reguli pe care nu

le cunosc *apriori*, ci le învață pe parcurs. Aceste reguli sunt aduse de *smart messages* care vor vizita fiecare nod. Este diferit de modelul coregrafiei deoarece participanții nu cunosc imaginea de ansamblu și nici nu controlează în totalitate răspunsul cererii pe care o primesc. Nu există un controller central (ca în modelul orchestrării).

4.8.6 Comunicarea într-un sistem de tip *swarm*

Comunicarea cu *swarms* se face folosind un limbaj descriptiv propriu. Comunicarea are loc între nodurile unui sistem distribuit. Un *nod* este definit ca orice entitate software capabilă să primească sau să trimită mesaje de tip *swarm*.

Există două tipuri de noduri:

- adaptor - noduri server care oferă diverse servicii, API-uri, *etc.*
- client - noduri care se conectează la adaptori *via* TCP

Fiecare nod este identificat printr-un nume - *nodeName* care poate lua următoarele valori cu următoarele specificații:

- *wellKnownNode* care sunt nodurile de infrastructură; un node de infrastructură este un nod care conține *core functionalities*
- *groups*, nume folosit pentru a grupa mai multe noduri; nodurile pot fi grupate de către un adaptor și sunt accesibile doar de cei care cunosc numele grupului
- *innerNodeName* care reprezintă un nod care folosește resursele unui nod normal; acesta nu are vizibilitate în afara nodului din care face parte și are rolul de a ajuta în faza de development (de exemplu, la compararea latențelor)

Etapele transmiterii unui mesaj folosind *swarm*-uri se pot observa în figura 16.

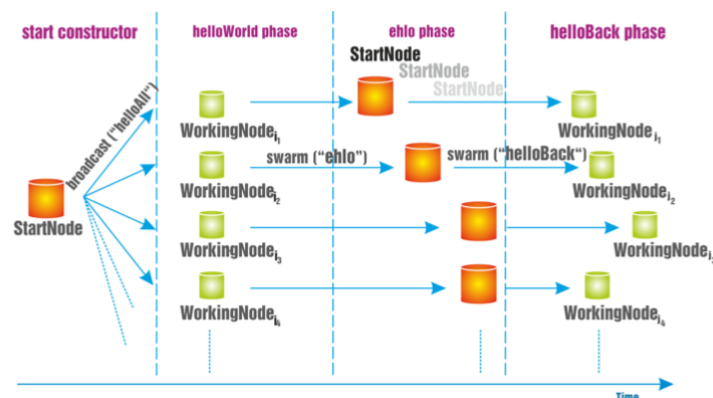


Figura 16: Etapele transmiterii unui "Hello World" folosind comunicarea cu swarms [24]

4.9 Private Data System

4.9.1 Imagine de ansamblu

Private Data System [29] este un sistem distribuit care permite stocarea și redistribuirea datelor private via internet. Modelul este construit astfel încât utilizatorii să fie în control total asupra datelor furnizate (*self-sovereign*). Sistemul este format din noduri distribuite care sunt folosite pe post de baze de date locale de tipul cheie-valoare (*key-value*). Comunicarea dintre noduri se face folosind coregrafii executabile (descrise mai sus în modelul *swarm*, secțiunea 4.8). Coregrafiile executabile sunt capabile să prevină scurgerile de informații în contextul comunicării *cross-organisation*.

Folosind PDS, utilizatorul deține în totalitate controlul asupra datelor sale și poate decide când și cu cine datele sale vor fi distribuite. De asemenea, utilizatorul poate revoca accesul oricui la orice moment. În cazul unui update (de exemplu, utilizatorul își schimbă domiciliul și trebuie să modifice adresa de livrare/facturare/etc.) datele vor fi propagate instant.

5 Coregrafii criptate

Modelul care urmează a fi prezentat este o colaborare în cadrul proiectului PrivateSky [30].

5.1 Descrierea modelului propus

Modelul propus vine ca o soluție pentru mitigarea atacurilor care rezultă în scurgeri de date, și constă în **criptarea datelor sensibile** și posibilitatea de a efectua operații pe aceste date criptate fără a fi nevoie să le decriptăm.

Unul din marile dezavantaje când vine vorba de lucrul cu date într-un mod sigur (folosind sisteme de criptare pentru a asigura confidențialitatea) este complexitatea calculului. Prin lucrul direct pe date criptate se elimină acest *overhead* adus de permanenta criptare și decriptare necesară filtrării datelor. O astfel de abordare permite clienților să creeze datele confidențiale la nivelul aplicației client și să nu dezvăluie cheile de criptare serverului de bază de date - și, implicit, nici administratorilor acestora. Se creează o separare între cei care dețin datele și cei care se ocupă de stocarea lor (și care nu ar trebui să aibă acces la datele în text clar). Oferind clienților beneficii de securitate *opt in* într-un mod ușor și cât se poate de transparent, aceștia pot crea aplicații mai sigure care să urmeze principiile “privacy by design” [31].

Coregrafiile criptate (*encrypted choreographies*) se bazează pe existența unor mijloace de control a cheilor de criptare și mecanisme de identificare și autentificare ce permit crearea de coregrafii sigure din punct de vedere al datelor criptate între două sau mai multe organizații. Coregrafiile verificabile își propun doar să reliefeze locul (organizația) și tipul de date private care se comunică, în timp ce coregrafiile criptate asigură un set de instrumente și o implementare în sine a coregrafiilor care are ca scop minimizarea partajării de informații private.

Sistemele inteligente moderne se bazează pe procese complexe controlate de sisteme de calcul. Din punct de vedere tehnic, perspectiva integrării este foarte importantă pentru sistemele inteligente. În cazul unui număr ridicat de puncte de integrare, procesul este rezolvat prin abordări consacrate de tipul ESB (*Enterprise Service Bus*), MOM (*Message-Oriented Middleware*), sisteme bazate pe EIP (*Enterprise Integration Patterns*), sau prin orchestrarea de servicii utilizând limbaje de programare sau cod adaptat modelării proceselor de business (*Business Process Modeling*).

Toate aceste metode sunt în general suficiente pentru integrarea elementelor aparținând unei singure organizații. Pe de altă parte, integrarea între organizații multiple ar trebui abordată utilizând coregrafii. Orice soluție centralizată este riscantă din punct de vedere al securității și protecției datelor personale.

Cu toate că multi autori în literatura de specialitate percep coregrafiile drept un mecanism limitat la descrierea formală a contractelor dintre un număr de organizații, cercetarea specifică mediului academic propune conceptul de coregrafii executabile (de exemplu, a se vedea descrierea implementării *SwarmESB* din secțiunea 4.8). Se sugerează astfel transcrierea descrierilor coregrafiilor în cod executabil de către fiecare organizație participantă la coregrafie. Astfel, coregrafia devine nu doar o descriere formală a unui contract dintre organizații, ci o descriere a unui proces operativ într-o formă executabilă. Aceeași secvență (coregrafie) va fi rulată de către mai multe organizații, astfel încât necesitatea traducerii coregrafiei în alte limbaje de programare dispare.

O coregrafie executabilă poate fi considerată o secvență de pași într-un proces de afaceri, proces (sau flux) care este rulat de către organizații multiple într-o manieră descentralizată. În timpul execuției, comunicarea dintre organizații se realizează prin mesaje asincrone, astfel încât secvența de procese operative este formată doar dintr-un set de etape care combină un stil declarativ pentru fazele executabile, cu un cod ce asigură un model de programare abordabil.

Implementarea coregrafiilor criptate se bazează pe existența unor sisteme de stocare a datelor ce folosesc tehnici de criptare specifice ce încearcă să realizeze implementări practice ale criptării homomorfe parțiale [10][32].

De asemenea, implementarea coregrafiilor criptate propune metode de stocare a datelor criptate între organizații independente unele de altele. Prin această metodă se urmărește dezvoltarea de protocoale criptografice care înainte de stocare, anonimizează și divizează datele, folosind coregrafii, într-un mod care minimizează riscul ca datele să fie copiate discreționar de un administrator sau atacator care controlează doar unul dintre nodurile (organizațiile) participante. Tot în contextul coregrafiilor encrypted putem menționa folosirea de protocoale criptografice pentru protejarea comunicării [33] și sistemele de modelare a politicilor de privacy [34].

5.1.1 Operații asupra datelor encrypted la nivelul ORM-urilor

Metoda propusă vrea să ofere un model inspirat din CryptDB [8] eliminând serverul proxy din arhitectura aplicației, operațiile de criptare fiind integrate la nivelul ORM-ului. Această schimbare nu doar că simplifică arhitectura, dar ușurează munca developerilor și reduce costurile de infrastructură. Serverul proxy poate fi privit ca un SPoF (Single Point of Failure). Dacă un atacator reușește să obțină acces la serverul proxy, acesta poate decripta toate datele stocate.

Modelul de securitate propus de noi este inspirat din CryptDB și funcționează în modul următor:

- Datele sensibile sunt descrise prin adnotarea la nivelul ORM-ului. În momentul creării ORM-ului se specifică operațiile care se doresc a fi suportate pe coloana respectivă. Coloanele care vor stoca date private sunt marcate prin adnotări. Un exemplu de adnotare făcută la nivelul ORM-ului se poate vedea în figura de mai jos. Cele două adnotări propuse sunt pentru cele mai uzuale operații: cea de egalitate care permite operații de tipul “=” și “!=” (‘eq’) și cea de ordine (‘ord’ - “<”, “>”, “<=”, etc).

```

name: {
    type: 'string',
    default: 'no name',
    pk: true,
    security: 'eq'
},
lastName: {
    type: 'string',
    default: 'no name',
    index: true
},
age: {
    type: 'int',
    default: 20,
    security: 'ord'
}

```

Figura 17: Exemplu de ORM creat cu **apersistence** care optează pentru beneficii de securitate

- Se utilizează un sistem extern de gestiune a cheilor de criptare. Pentru asta propunem folosirea unui sistem similar PDS-ului propus în lucrarea [29] prin care se pastrează cheile de criptare conform rolurilor utilizatorilor care au acces la date și nu generate prin derivarea cheilor de criptare din parola utilizatorului.
- Accesul la cheile de decriptare se face prin PDS conform regulilor de acces din PDS. PDS oferă un model de stocare a datelor sensibile folosind baze de date distribuite. Datele sunt împărțite în chunk-uri indescifrabile și distribuite folosind *executable choreographies*.

5.1.2 Interacțiunea dintre ORM și PDS

ORM-ul va folosi un sistem de tip PDS pentru a stoca informații private (de exemplu parole de criptare pentru datele expuse). Pentru a formaliza comunicarea dintre ORM și PDS, se va utiliza o versiune îmbunătățită a notațiilor originale din lucrarea care propune sistemul PDS [29].

Se identifică următoarele **categorii de informații**:

- **DO (Data Owner)** - informații despre proprietarul informației

- **PD (Private Data)** – informația personală care va fi stocată în sistem; dacă o secvență de informație personală p este împărțită în n calupuri indescifrabile, atunci $p_i, i \in 0 \dots n$ este un calup indescifrabil de informații
- **MD (Metadata)** – descrie relația dintre Informația Personală și Operatorul Informației prin etichetarea informației în conformitate cu Proprietarul Informației

Având în vedere că sistemul se bazează pe baze de date de tip cheie-valoare, vom defini următoarele **chei pentru stocarea, asocierea și referința informațiilor**:

- **MK (Master Key)** – *Cheie Primară*; reprezintă o secvență de informație personală; anonimizează o secvență de informație personală
- **PK (Partial Key)** – *Cheie Parțială*; reprezintă și anonimizează un calup indescifrabil dintr-un set de calupuri indescifrabile în care a fost împărțită o secvență de informație personală; astfel, cheia principală este asociată unui set de chei parțiale care reprezintă un set de calupuri indescifrabile necesare recompunerii secvenței de informație personală
- **KR (Key Reference)** – *Cheie de Referință*; reprezintă o referință la / un alias al unei secvențe de informație personală (o referință către o Cheie Principală)
- **KRH (Key Reference Hash)** – *Cheie de Referință Hash*; se obține prin aplicarea unei funcții hash pe o valoare a Cheii de Referință, și prin adăugarea adresei nodului de procesare care va primi rezultatul

Se definesc următoarele **categorii de noduri** în PDS:

- **PN (Processing Node)** – *Nod de Procesare*; stochează Cheile de Referință și trebuie să primească și proceseze informația personală la care fac trimitere acestea. Nodurilor de Procesare le este interzis să stocheze informația primită pe termen lung. În cadrul propunerii noastre PN va fi notat și ORM pentru a pune mai ușor în evidență locul unde se află ORM-ul.
- **AN (Audit Node)** – *Nod Audit*; gestionează o bază de date ce conține asocierile dintre Cheile de Referință și Cheile Principale la care acestea fac trimitere, concomitent cu descrierea informației la care face trimitere Cheia Principală (Proprietarul Informației, Operatorul Informației și Metadata). În această bază de date, cheia este Cheia de Referință, în timp ce valoarea este o tuplă conținând Cheia Principală, Metadata, Proprietarul Informației și Operatorul Informației.

- **IN (Index Node)** – *Nod Index*; gestionează o bază de date care stochează asocierile dintre Cheile Principale și Cheile Parțiale corespunzătoare. În baza de date, cheia este Cheia Principală, în timp ce valoarea este o listă de Chei Parțiale necesare reconstituirii secvenței de informație personală reprezentată de către Cheia Principală
- **SN (Storage Node)** – *Nod de Stocare*; gestionează o bază de date care stochează asocierile dintre Cheile Parțiale și Mesajele Parțiale. În această bază de date, cheia este Cheia Parțială, în timp ce valoarea este cuplul indescifrabil de informație reprezentat de o anumită Cheie Parțială

Modul în care se poate obține o parolă de criptare din PDS este reprezentat în *figura 18*

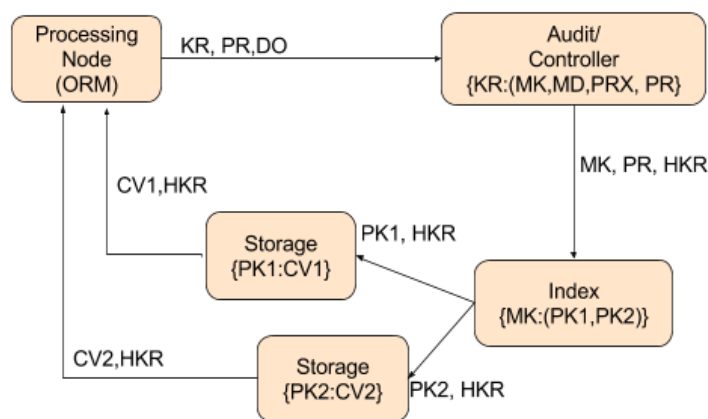


Figura 18: Modul de obținere a unei parole din PDS folosind 2 noduri storage

Pentru a formaliza operațiile cu PDS-ul vom folosi următoarele notații:

- $[E_1, E_2, \dots, E_n]$ reprezintă o listă conținând elementele E_1, E_2, \dots, E_n
- (E_1, E_2, \dots, E_n) reprezintă o tuplă conținând elementele E_1, E_2, \dots, E_n
- $\{K_1 : V_1, K_2 : V_2, \dots, K_n : V_n\}$ reprezintă un dicționar în care valoarea V_1 este stocată sub cheia K_1 , valoarea V_2 este stocată sub cheia K_2 , ..., iar valoarea V_n este stocată sub cheia K_n
- $N_1 \rightarrow N_2 : M$ înseamnă că nodul N_1 trimite nodului N_2 mesajul M , operație care corespunde executării unui pas în coregrafia executabilă
- $DB[K] := V$ înseamnă că valoarea V este stocată sub cheia K în baza de date DB de către nodul ce gestionează DB
- $V := DB[K]$ înseamnă că valoarea V asociată cheii K este obținută din baza de date DB de către nodul ce gestionează DB

- $N_1 : A$ înseamnă că nodul $N1$ execută acțiunea A
- $M := gen()$ înseamnă că mesajul M este generat (fie el aleator sau conform unui algoritm); aceasta reprezintă o acțiune
- $PD_1, PD_2, \dots, PD_n := split(PD)$ înseamnă că data personală PD este împărțită în n calupuri indescifrabile de informații PD_1, PD_2, \dots, PD_n ; aceasta reprezintă o acțiune
- $PD := recombine(PD_1, PD_2, \dots, PD_n)$ înseamnă că n calupuri indescifrabile de informații PD_1, PD_2, \dots, PD_n sunt recombuse pentru a crea informația personală PD inițială care a fost împărțită pentru a le crea; aceasta reprezintă o acțiune

Se prezintă mai jos modul de interacțiune dintre ORM și componentele PDS-ului pentru a obține parolele de criptare.

Etapa 1 - trimite către sistemul PDS identificat de un nod de audit (AN) un KR pentru parolă

1. $ORM \rightarrow AN : ORM, KR, DO$

Etapa 2 - AN determină nodurile unde informația e stocată

1. $MK := AN[KR]$
2. $HKR := (location(PN), hash(KR))$
3. $AN \rightarrow IN : ORM, MK, HKR$

Etapa 3 - informația (bucăți din parolă) este transmisă la ORM din fiecare nod

1. $PK_1, PK_2, \dots, PK_n := IN[MK]$
2. $IN \rightarrow SN_i : ORM, HKR, PK_i, i \in \{1..n\}$
3. $PD_i := SN_i[PK_i], i \in \{1..n\}$
4. $SN_i \rightarrow ORM : HKR, PD_i, i \in \{1..n\}$

Etapa 4 - parola este refăcută în ORM și folosită în operații

1. $ORM : PD := recombine(PD_1, PD_2, \dots, PD_n)$, unde $PD_i, i \in \{1..n\}$ trebuie să aibă aceeași HKR ca și PD

Dacă un nod de procesare care încapsulează ORM-urile trebuie să decripteze o informație personală, trebuie să cunoască cheie de referință corespunzătoare parolei necesare pentru decriptare (KR). Trimițând cheia sa de referință nodului audit, împreună cu identitatea sa (a nodului de procesare), nodul de procesare încheie **prima etapă**.

În a **doua etapă**, nodul audit obține cheia principală corespunzătoare cheii de referință primite. În continuare, rulează HKR , care reprezintă o funcție hash aplicată cheii de referință obținute, prefixată cu locația nodului de procesare. Nodul audit trimite apoi nodului index identitatea nodului de procesare, cheia principală obținută și HKR calculat la pasul anterior. Astfel, nodul index nu află asocierile dintre cheile de referință și cheia principală, însă în același timp distribuie rezumatul hash, informație cerută de către nodul de procesare pentru a identifica solicitarea căreia trebuie să îi răspundă. Se observă că un nod de procesare dat poate rula mai multe operațiuni de obținere de informații simultan și, fără HKR , nodul de procesare nu poate ști cărui calup indescifrabil îi corespunde o secvență de informație personale pe care a solicitat-o simultan.

În a **treia etapă**, nodul index obține din baza sa de date cheile parțiale corespunzând cheii principale și trimite fiecare cheie parțială împreună cu identitatea nodului de procesare și HKR către nodurile de stocare corespunzătoare. Fiecare nod de stocare obține valoare indescifrabilă (PD_i), corespunzând cheii parțiale primite (PK_i) și trimite nodului de procesare calupul indescifrabil obținut și HKR . Nodul de procesare, după primirea calupurilor indescifrabile, le grupează după HKR și apoi reconstituie elementele grupate pentru a obține informația personală. Parola poate acum fi recuperată (**pasul 4**), însă este interzis nodului de procesare să o stocheze.

Scopul HKR este de identificator, astfel încât un nod de procesare care primește secvențe multiple de informații private simultan să poată asocia calupurile indescifrabile de informație cu cheile de referință solicitate. Stocarea cheilor KR necesare identificării în PDS a parolelor se poate face direct în câmpurile din baza de date unde se stochează informația criptată. Sistemul de autentificare și autorizare din PDS poate să prevină atacuri din interior deoarece va refuza să rezolve cereri de la utilizatori care nu sunt autentificați. Această soluție constituie o îmbunătățire față de soluția derivării parolelor de criptare din parolele de autentificare a utilizatorilor deoarece devine mai dificil pentru un atacator să folosească datele criptate în afara operațiilor normale ale sistemului.

5.2 Avantajele modelului

- Unul dintre cele mai mari impedimente când vine vorba managementul parolelor este re-distribuirea acestora în toate locurile necesare în momentul schimbării acesteia. Din motive de securitate, utilizatorii ar trebui încurajați ca la un anumit interval de timp să își schimbe parola (cum este exemplu aplicațiilor de homebanking unde odată la 3 luni trebuie să îți schimbi parola, altfel nemaiprimind acces la detaliile contului). De asemenea, parolele sunt predispuse la erori umane. Cei de la PasswordResearch [35] susțin că 82% din oameni si-au uitat parola cel puțin o dată. În modelul propus schimbarea parolei utilizatorului nu afectează datele deja criptate, așadar overhead-ul re-securizării datelor fiind minimizat.
- În securitatea informației, RBAC (Role Based Access Control) este un model care presupune acordarea privilegiilor asupra unor resurse în funcție de rolul utilizatorului. Modelul de față este conceput pentru a funcționa într-un mediu multi-user complex în care utilizatorii au diferite roluri (role based) și li se acordă drepturi de citire asupra datelor în funcție de acestea.
- Se pot folosi parole unice pentru fiecare din datele criptate (la nivel de model/tabel sau coloană)
- Cheile de criptare sunt distribuite în chunk-uri nedescifrabile, ceea ce înseamnă că dacă un atacator reușește să afle cheia de criptare a unui utilizator nu implică aflarea cheilor tuturor utilizatorilor. Blast radius-ul atacului este limitat la aflarea datelor la care are acces utilizatorul compromis.

5.3 Moduri de atac

- Din moment ce datele sunt criptate pe partea de client, folosind parola utilizatorului, pentru a afla datele confidențiale este nevoie de cheile de criptare. Cheile de criptare pot fi obținute de un intrus prin atacarea PDS-ului. Pentru a mitiga impactul atacării PDS-ului, acesta poate fi securizat cu sisteme automate de prevenție ce învață *pattern*-uri de folosire a datelor [36].
- Un risc pe care îl aduc sistemele role-based este cel al escalării privilegiilor (privilege escalation). Prin escalarea rolului său, un atacator ar putea obține informații la care nu ar trebui să aibă acces.

- Din moment ce un sistem de criptare care păstrează ordinea dezvăluie niște detalii despre natura datelor (cum ar fi distribuția lor). Un atacator cu acces la datele stocate în baza de date poate afla diverse metainformații (numărul de valori distincte, cum sunt comparabile două valori, etc).

6 Aplicație PoC folosind coregrafii encrypted

După cum am menționat și în secțiunea de avantaje ale modelului coregrafiilor encrypted, acestea pot fi folosite pentru a crea aplicații **role-based**. În secțiunile ce urmează se va prezenta o astfel de aplicație *proof-of-concept*, folosind coregrafii encrypted.

6.1 Use case

Se poate observa în ziua de astăzi o continuă creștere a centralizării informațiilor în medii online. În era tehnologiei oamenii își doresc să găsească orice informație de interes la doar un click distanță, cât mai repede posibil. Și această dorință pentru unii este o necesitate. Pentru marile instituții (cum ar fi cele de învățământ) sau companiile cu un număr crescut de angajați devine anevoioasă menținerea centralizată a tuturor datelor de interes și facilitarea unui răspuns rapid, la cerere.

Angajații au diverse necesități, și în lipsa unor sisteme accesibile online care să le ofere informațiile dorite, aceștia sunt nevoiți să caute răspunsuri în alte părți.

- *Câte zile de concediu mai am anul ăsta?*
- *Cât mi-a crescut salariul la ultima promovare?*
- *Care este numărul de telefon al colegului nou?*
- *Care este data la care m-am angajat?*
- *Care este numele de familie al lui Dan?*
- *etc.*

Aplicația de față vrea să arate că întrebări cum ar fi cele de mai sus își pot găsi răspunsul într-o singură aplicație, oferind acces pe baza de roluri la date. Datele private cât și cele publice sunt oferite de o interfață comună în loc de aplicații diferite pentru fiecare nivel de privacy al datelor.

Aplicația propusă răspunde minimalist la două întrebări: *Care este salariul meu la momentul actual?* și *Care este numărul de telefonul al colegului meu și ce informații mai pot afla despre el în scurt timp?*

Informațiile despre salariul sunt confidențiale la nivel de individ. Descrierea personală și numărul de telefon sunt considerate confidențiale la nivel de departament.

6.2 Arhitectura aplicației

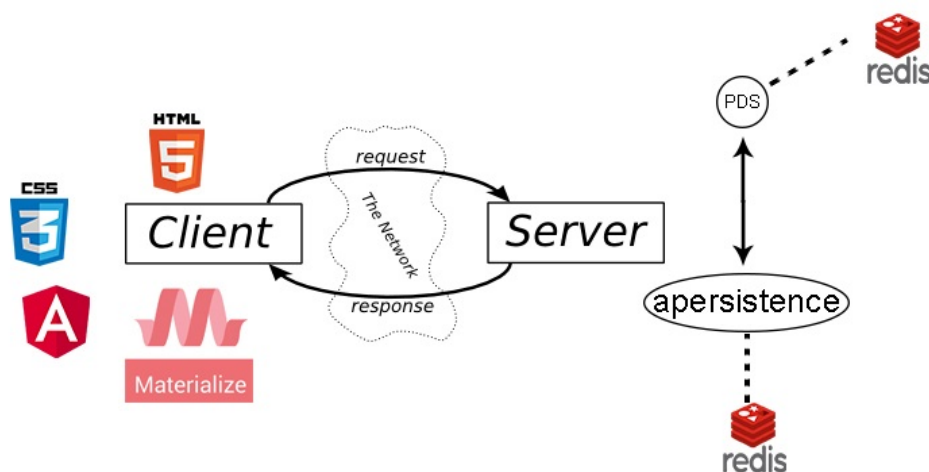


Figura 19: Arhitectura aplicației PoC folosind coregrafii encrypted.

6.2.1 Apersistence

Secțiunea de față prezintă îmbunătățirile de securitate aduse modului *apersistence* descris în secțiunea 4.7. Acest modul este folosit într-un context mai larg, cel al proiectului *SwarmESB*, a cărui paradigmă și arhitectură este descrisă în secțiunea 4.8.

Inițial nu exista posibilitatea ca datele să fie criptate înainte de a fi trimise între noduri. Adăugările sunt făcute la nivelul adnotărilor, la crearea modelului de persistență, ulterior devenind transparente developerilor.

Modelul de securitate adăugat este inspirat din CryptDB, descris în secțiunea 3.1.

Figurile 21 și 20 compară același obiect creat cu *apersistence*, unul folosind adăugările de securitate, celalalt nu.

```
{ __meta:
  { typeName: 'Person',
    freshRawObject: true,
    savedValues: { name: 'ana',
                  lastName: 'popescu' },
    getPK: [Function: bound ],
    getPKField: [Function],
    loadLazyField: [Function],
    loadLazyFields: [Function] },|
  assign: [Function: bound castAssign],
  name: 'ana',
  lastName: 'popescu' }
```

Figura 20: Înregistrare creată fără a folosi beneficiile de securitate din *apersistence*

Se poate observa cum în figura 20 datele sunt în text clar, în timp ce în figura 21 acestea sunt

în text criptat. Astfel, un atacator care interceptează datele după ce acestea au fost criptate și nu are acces la cheia de criptare nu poate fura informațiile transmise.

```
{ __meta:
  { typeName: 'Person',
    freshRawObject: true,
    savedValues:
      { name: 'f458cb536c4eb9ee749dbaa3e67144fc',
        lastName: '0ce8612d92e708fb5247235378b37582' },
    getPK: [Function: bound ],
    getPKField: [Function],
    loadLazyField: [Function],
    loadLazyFields: [Function] },
  assign: [Function: bound castAssign],
  name: 'f458cb536c4eb9ee749dbaa3e67144fc',
  lastName: '0ce8612d92e708fb5247235378b37582' }
```

Figura 21: Înregistrare creată folosind caracteristicile de securitate din *apersistence*

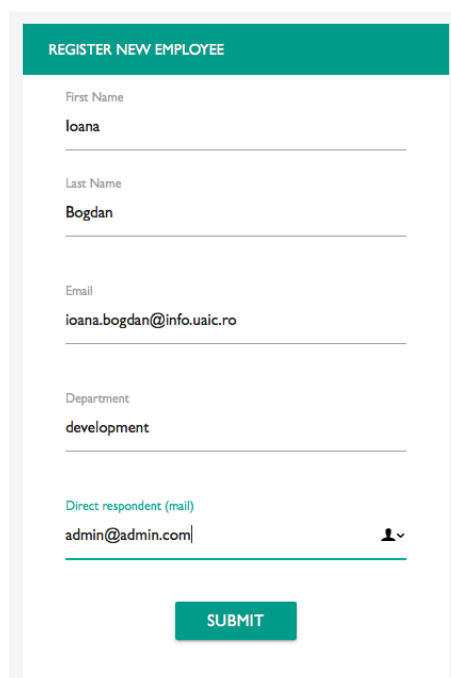
6.2.2 PDS

Pentru acest *proof of concept* s-a implementat minimal un *Private Data System* după descrierile din secțiunile dedicate descrierii interacțiunii cu acest sistem (4.9 și 5.1.2).

6.3 Funcționalități oferite

6.3.1 Login & Register

Pentru a înregistra un utilizator în aplicație se va folosi un administrator. Acesta ca introduce datele despre angajat după cum se poate observa în figura 22.



REGISTER NEW EMPLOYEE

First Name
Ioana

Last Name
Bogdan

Email
ioana.bogdan@info.uaic.ro

Department
development

Direct respondent (mail)
admin@admin.com

SUBMIT

Figura 22: Pagina de *register*

În momentul înregistrării unui nou angajat acestuia i se atribuie o parola *default* pe care ulterior, în momentul primei accesări a aplicației, utilizatorul va trebui să o schimbe.

Parolele sunt gestionate folosind *Private Data System* descris în secțiunile 4.9 și 5.1.2.

Formularele de *login* și *change password* arată similar celui de *register*.

6.3.2 Pagina personală

Pagina personală cuprinde minimal detalii despre utilizatorul logat la momentul respectiv. Informațiile din câmpurile *info* și *phone* sunt partajate la nivel de **departament**. Salariul este partajat la nivel de **indiv** (doar utilizatorul are acces la salariul său).

The screenshot shows a user profile page with a teal header containing three tabs: 'PROFILE', 'COLLEAGUES', and 'LOGOUT'. The 'PROFILE' tab is active. Below the header, there are several input fields with labels and values: 'First name:' with 'Ioana', 'Last name:' with 'Bogdan', 'Info:' with a long Lorem Ipsum text, 'Phone:' with '+40749491404', 'Department:' with 'development', 'Email:' with 'ioana.bogdan@info.uaic.ro', 'Direct Respondant:' with 'admin@admin.com', and 'Salary:' with '392'. At the bottom left of the form is a teal 'EDIT' button.

Figura 23: Profilul personal

În momentul în care utilizatorul navighează pe profilul unui angajat din alt departament, acesta nu va avea acces decât la datele publice (nume, prenume, adresa de email, departament și cine este *direct respondent*).

The screenshot shows a user profile page for a user from a different department. The fields are: 'First name:' with 'Sancho', 'Last name:' with 'Bodocs', 'Info:' with 'error', 'Phone:' with 'error', 'Department:' with 'research', 'Email:' with 'sbodocs0@sbodocs0.com', 'Direct Respondant:' with 'admin@admin.com', and 'Salary:' with 'error'. The 'Salary' field is highlighted with a red border, indicating an error.

Figura 24: Profilul unui angajat din alt departament. Câmpurile pe care nu le poate accesa utilizatorul curent au fost lăsate intenționat pentru a arăta informația primită de la server.

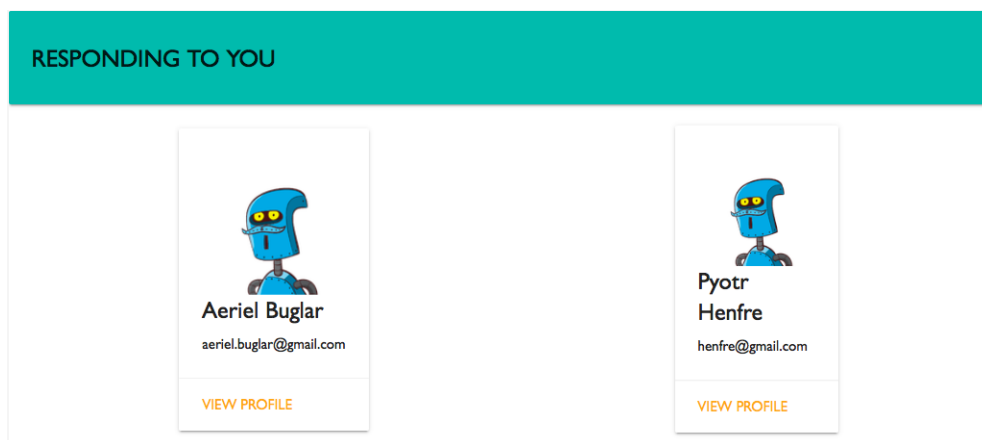
Cât despre datele stocate, în figura 25 se poate observa cum datele menționate ca private sunt păstrate criptat.

```
{
  \"email\": \"ioana.bogdan@info.uaic.ro\",
  \"name\": \"Ioana \",
  \"lastname\": \"Bogdan\",
  \"info\": \"29402774bb1a70886c893fdc50494129cf6c55e9f561a1edab0dd3aaa057[...]\",
  \"salary\": \"efd95c8dc625a483e5f05615342891b8\",
  \"directRespondant\": \"admin@admin.com\",
  \"phone\": \"d976bb26459ace5a410962c6aad60cc3\",
  \"department\": \"development\",
  \"profilePic\": \"images/default.gif\", \"files\": \"\"
}
```

Figura 25: Exemplu de date ale utilizatorului care sunt stocate criptate în baza de date.

6.3.3 Echipa mea


Utilizatorul poate de asemenea naviga prin structura ierarhică a companiei sau instituției în care lucrează prin intermediul paginii de colegi, unde poate vedea cine este direct respondent-ul lui, colegii subordonați aceluiași om cât și cei care îi raportează direct.



7 Contribuții

7.1 Submisie ICCP

Modelul coregafiilor encrypted prezentat în lucrarea de față este propus la **2017 IEEE 13th International Conference on Intelligent Computer Communication and Processing** [37] de către Lenuța Alboaie, Sînică Alboaie și autoarea lucrării de față.

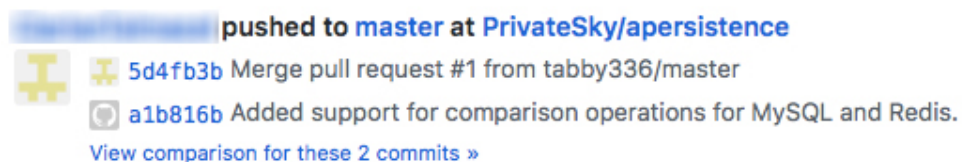
Paper 83	
Title:	Operations on encrypted data in an ORM made for encrypted choreographies
Paper:	
Author keywords:	encrypted database encrypted choreography executable choreography Private Data System
EasyChair keyphrases:	encryption key (160), processing node (140), encrypted data (110), executable choreography (100), database server (90), personal data (90), sensitive data (80), master key (80), user password (70), private data (70), application server (60), unreadable chunk (60), encrypted choreography (60), privilege escalation (50), partial key (50), acronym acceptable line (47), private data system (47), breach level index (47), audit node (40), adjustable query based encryption (40), index node (40), data breach (40), storage node (40), encryption layer (40), dynamic service oriented architecture (40), reference key (40), encryption password (40), private information (40), proxy server (40)
Topics:	Distributed Computing and Networking
Abstract:	It is estimated that, nowadays, around 40 records containing private data are leaked every second. Many of these illegally obtained records are due to the lack of security measures taken into consideration. One of the biggest mistakes that can be done, when it comes to data security, is to store data in plaintext, because a vulnerability that gives an attacker access to the database may result in a big data breach. This article's proposal is to mitigate, in a unobtrusive way, the threat of exploiting databases described above by adding cryptographic measures to sensitive data, ensuring that a snoopy database administrator, or even an outsider that gained access to the storage server, can't reveal private information even if they have it. With the time complexity problem in mind, as databases are selling on processing speed, our proposal is a practical solution to execute a suite of queries on encrypted data directly on the server. Our contribution consists in adding this cryptographic facilities directly in the ORM (Object Relational Mapping) and a new method for password administration.
Time:	Jun 17, 07:45 GMT

Authors						
first name	last name	email	country	organization	Web page	corresponding?
Alboaie	Sînică	salboaie@gmail.com	Romania	Faculty of Computer Science Alexandru Ioan Cuza University of Iasi		✓
Ioana	Bogdan	ioana.bogdan@info.uaic.ro	Romania	Faculty of Computer Science Alexandru Ioan Cuza University of Iasi		
Alboaie	Lenuța	adria@info.uaic.ro	Romania	Faculty of Computer Science Alexandru Ioan Cuza University of Iasi		✓

7.2 Contribuții *open source*

Printre operațiile de criptare propuse este cea de *order preserving encryption* care aduce beneficiul executării rapide a interogărilor pe anumite intervale (*range query*). Pentru acest tip de interogări este necesar să se suporte operații de comparare (" $<$ ", " $>$ ", " $!=$ ", etc) pe care *apersistence* nu le suporta la momentul respectiv.

Contribuția prezentată în lucrarea de față asupra modulului *open source apersistence* constă în implementarea filtrelor pentru operații de comparație atât pentru valori numerice, cât și pentru șiruri de caractere.



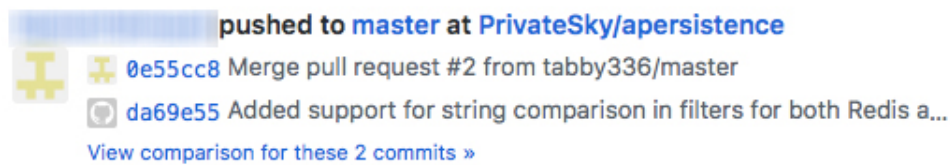


Figura 26: Contribuțiile *open source* în modulul *apersistence*

Pentru implementarea operațiilor de comparație au fost necesară adăugări atât în logica de parsare a filtrelor, cât în cea de obținere a rezultatelor din baza de date. De asemenea, au mai fost modificate testele unitare. Adăugările menționate sunt atât în ORM-ul pentru Redis cât și în cel pentru MySQL.

Schimbările nu au putut fi făcute într-un singur nivel de abstracție pentru ambele baze de date deoarece **Redis** nu are suport nativ pentru *range query*, logica de filtrare trebuind construită diferit față de cea pentru MySQL. Pentru a obține valorile dintr-un anumit interval pentru o coloană indexată mai întâi se preiau din baza de date doar cheile pentru coloana respectivă, se filtrează și ulterior se preia și valoarea pentru fiecare cheie care satisface condiția din filtre. Pentru **MySQL** s-a construit clauza *WHERE* corespunzător.

```
var rawData = [
  {id: "2", name: "Dana", location: "Tecuci",sex:true, age:25},
  {id: "3", name: "Ana", location: "Iasi",sex:true, age:21},
  {id: "4", name: "Ana", location: "Bucuresti",sex:true, age:21},
  {id: "5", name: "Ion", location: "Iasi",sex:false, age:24}
];

var filterTests = [
  {
    modelName:modelName,
    filter:{name:"!=Ana", age:"<25"},
    expectedResults: [{id: 5, name: "Ion", location: "Iasi",sex:false, age:24}]}
];
```

Figura 27: Exemplu de filtre adăugate în *apersistence* și rezultatul așteptat pentru acestea.

8 Concluzie

În această lucrare a fost prezentat un model practic prin care sistemele informatice pot permite interogări direct pe date criptate. Aceste interogări se procesează la nivelul serverului bazei de date. Astfel, se micșorează suprafața de atac asupra datelor sensibile stocate în bazele de date. Din considerente practice, unele operații necesită decriptarea datelor, dar riscul cel mai mare pentru pierderea datelor este constituit de stocarea acestora în text clar în bazele de date. Față de alte abordări existente, modelul de față constă în modificarea unui ORM pentru a suporta operații de criptare, și în modul de gestiune a parolelor de criptare.

9 Bibliografie

- [1] Ian Foster et all. “Cloud Computing and Grid Computing 360-Degree Compared”. In: (2008). doi: <https://arxiv.org/pdf/0901.0131.pdf>.
- [2] *Internet users have average of 5.54 social media accounts*. URL: <http://blog.globalwebindex.net/chart-of-the-day/internet-users-have-average-of-5-54-social-media-accounts/>.
- [3] *Online ID OD: illegal web trade in personal information soars*. URL: <https://www.experianplc.com/media/news/2012/illegal-web-trade-in-personal-information-soars/>.
- [4] *Breachlevelindex*. URL: <http://breachlevelindex.com>.
- [5] *Breach Level Index Report 2016 Gemalto*. URL: <http://breachlevelindex.com/assets/Breach-Level-Index-Report-2016-Gemalto.pdf>.
- [6] İhsan Haluk Akin and Berk Sunar. “On the Difficulty of Securing Web Applications using CryptDB”. In: (2015). doi: <https://eprint.iacr.org/2015/082>.
- [7] Raluca Ada Popa, Nickolai Zeldovich, and Hari Balakrishnan. “Guidelines for Using the CryptDB System Securely”. In: (2015). doi: <http://eprint.iacr.org/2015/979>.
- [8] Raluca Ada Popa et al. “CryptDB: Protecting Confidentiality with Encrypted Query Processing”. In: (2011). doi: <http://people.csail.mit.edu/nickolai/papers/raluca-cryptdb.pdf>.
- [9] A. Boldyreva et al. “Order-preserving symmetric encryption”. In: (2009). doi: <https://pdfs.semanticscholar.org/f75c/5ee8db682f37f0ebf16d62677823c02359af.pdf>.
- [10] Yi Xun, Paulet Russell, and Bertino Elisa. *Homomorphic Encryption and Applications*. 2014. Chap. 2, Homomorphic Encryption.
- [11] Dawn Xiaodong, Song David, and Wagner Adrian Perrig. “Practical Techniques for Searches on Encrypted Data”. In: (2000). doi: <http://dl.acm.org/citation.cfm?id=884426>.
- [12] *BigQuery Encrypted Client*. URL: <https://github.com/google/encrypted-bigquery-client>.

- [13] *Microsoft Always Encrypted*. URL: <https://docs.microsoft.com/en-us/sql/relational-databases/security/encryption/always-encrypted-database-engine>.
- [14] Craig Gentry. “Fully Homomorphic Encryption Using Ideal Lattices”. In: (). doi: <https://www.cs.cmu.edu/~odonnell/hits09/gentry-homomorphic-encryption.pdf>.
- [15] *Relational databases*. URL: https://en.wikipedia.org/wiki/Relational_database.
- [16] *Jumping from SQL to Firebase NoSQL Database*. URL: <https://gonehybrid.com/firebase-database-best-practices/>.
- [17] Christof Strauch. *NoSQL Databases*. 2009. Chap. 2, The NoSQL-Movement - Motives of NoSQL practioners.
- [18] *Redis Database*. URL: <https://redis.io/topics/introduction>.
- [19] *Redis*. URL: <https://aws.amazon.com/elasticache/redis/>.
- [20] *npm*. URL: <https://www.npmjs.com/>.
- [21] *ORM definition*. URL: <http://searchwindevelopment.techtarget.com/definition/object-relational-mapping>.
- [22] *ActiveAndroid Guide ORM*. URL: <https://guides.codepath.com/android/activeandroid-guide>.
- [23] *apersistence npm package*. URL: <https://www.npmjs.com/package/apersistence>.
- [24] Lenuța Alboaie, Sînică Alboaie, and Andrei Panu. “Swarm Communication - A Messaging Pattern Proposal for Dynamic Scalability in Cloud”. In: (2014). doi: <http://ieeexplore.ieee.org/document/6832160/>.
- [25] *Service-oriented architecture*. URL: https://en.wikipedia.org/wiki/Service-oriented_architecture.
- [26] *What is application orchestration?* URL: <https://www.mulesoft.com/resources/esb/what-application-orchestration>.
- [27] *Orchestration vs. Choreography*. URL: <https://stackoverflow.com/questions/4127241/orchestration-vs-choreography>.
- [28] *SwarmESB*. URL: <https://github.com/salboaie/SwarmESB>.
- [29] Sînică Alboaie and Doina Cosovan. “Private Data System Enabling Self-Sovereign Storage Managed by Executable Choreographies; DAIS 2017”. In: ().

- [30] *PrivateSky*. URL: <https://profs.info.uaic.ro/~ads/PrivateSky>.
- [31] *Ross McKean. EU data protection reform - privacy-by-design*. URL: <http://www.olswang.com>.
- [32] U. et. all Erlingsson. “RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response”. In: (2014).
- [33] *Tor Project*. URL: <https://www.torproject.org>.
- [34] A. Cavoukian and D. Jutla. “Privacy Policies Are Not Enough: We Need Software Transparency”. In: (2014).
- [35] *Attitudes and Behavior Towards Password Use on the World Wide Web*. URL: <http://passwordresearch.com/stats/statistic97.html>.
- [36] *System predicts 85 percent of cyber-attacks using input from human experts*. URL: <http://news.mit.edu/2016/ai-system-predicts-85-percent-cyber-attacks-using-input-human-experts-0418>.
- [37] *ICCP*. URL: <http://www.iccp.ro/iccp2017>.

Appendices

A Setarea mediului de lucru

Lucrarea de față a fost dezvoltată pe *macOS Sierra*. Comenzile pentru instalarea *npm* s-ar putea să nu funcționeze pentru alte distribuții de UNIX.

A.1 Instalare *Node Package Manager*

Terminal

```
brew install node
```

A.2 Instalare *apersistence*

Pentru a instala *npm* este nevoie în prealabil de XCode și homebrew.

1. Descărcarea modului *apersistence* de pe GitHub

Terminal

```
git clone https://github.com/PrivateSky/apersistence
```

2. Instalarea dependențelor din *package.json*

Terminal

```
npm install
```

A.3 Instalare *redis*

Terminal

```
~ wget http://download.redis.io/redis-stable.tar.gz
~ tar xvzf redis-stable.tar.gz
~ cd redis-stable
~ make
```