TODO

- Undergraduate Research Opportunities -

Dragoş Alin Rotaru
Mentor: Ruxandra F. Olimid

Universitatea din Bucureşti
ROMANIA
r.dragos0@gmail.com

Abstract. TODO - abstract

Cuprins

1 Introducere

1.1 Istoric

Termenul de criptografie este definit in dictionarul Oxford ca fiind "arta de a scrie si a rezolva coduri". Criptografia moderna s-a desprins de cea clasica in jurul anilor '80, motivand implementarea rigurozitatii matematice pentru definirea constructiilor criptografice. Asta pentru ca in anii anteriori, experienta a dovedit nesiguranta metodelor de criptare, criptanaliza lor fiind uneori triviala (cifrul lui Cezar, Vigenere [1], [2]) sau uneori atinsa cu ceva mai mult efort precum Enigma si alte metode din cel de-al doilea razboi mondial. **??**

Criptografia moderna se gaseste pretutindeni in viata de zi cu zi de la ATM-uri, cartele telefonice la semnaturi digitale, protocoale de autentificare, licitatii electronice sau bani digitali, luand amploare o data cu aparitia sistemelor cu cheie publica. O definitie potrivita ar fi "studiul stiintific al tehnicililor pentru a securiza informatia digitala, tranzactiile si calculul distribuit.". [3]

1.2 Motivatie

TODO nu are legătură: faptul ca sunt scheme cuantice nu are impact asupra structurii de acces; e ok sa menţionezi, dar atunci faci asta în secţiunea anterioară, când poţi să adaugi şi alte modalităţi de definire: scheme bazate pe latici, scheme bazate pe perechi biliniare, scheme cuantice, etc. Dezavantajul schemelor generale de partajare este dimensiunea componentelor, exponentiala in functie de numarul de participanti. [4] De asemenea, s-au dezvoltat scheme pentru modele de calcul neconventional, cum ar fi cel cuantic. [5]

1.3 Structura

TODO

1.4 Securitatea Teoretica a Informatiei

In cazul unor criptosisteme acestea nu pot fi compromise chiar daca adversarul dispune de o putere computationala nelimitata. Cateva exemple de criptosisteme care garanteaza securitatea teoretica-informationala sunt: schemele de partajare, unele protocoale multi-party computation, preluarea intr-un mod sigur(securizat?) informatii de la baze de date. Securitatea teoretica vine insa cu un cost: efortul computational depus este mult mai mare decat in cazul schemelor care nu garanteaza securitatea teoretica (se bazeaza pe dificultatea computationala unor probleme cunoscute). [6]

2 Scheme de partajare

O schemă de partajare constă în distribuirea unei informaţii secrete $\mathcal{S}$ la mai mulţi participanţi $\mathcal{P} = \{P_1, \ldots, P_n\}$ astfel încât oricare mulţime de participanţi predefinită ca făcând parte dintr-o structură de acces pe care o vom denumi $\mathcal{A}$ să poată reconstitui secretul $\mathcal{S}$. Formal, o schemă de partajare este reprezentată de o pereche de algoritmi (Gen, Rec):

- $Gen(S, m)$ este un algoritm care primeşte la intrare un secret S şi un număr întreg m şi întoarce un set de componente $s_1, s_2, \ldots, s_m$.
- $Rec(s_{i1}, s_{i2}, \ldots, s_{iq})$ este un algoritm care primeşte ca parametri de intrare o mulţime de componente şi întoarce S dacă mulţimea $\{P_{i1}, P_{i2}, \ldots, P_{iq}\} \in \mathcal{A}$.

Majoritatea schemelor constau în mai multe etape precum:

- *Iniţializare*. Presupune iniţializarea variabilelor de mediu necesare.
- *Generare*. O entitate autorizată (numită dealer) $\mathcal{D}$ foloseşte algoritmul *Gen* pentru a genera componentele.
- *Distribuţie*. Componentele sunt trimise participanţilor cu ajutorul unui mijloc de comunicare sigur, fără ca acestea sa fie vizibile unui atacator.
- *Reconstrucţie*. Dându-se o mulţime de componente, se foloseşte algoritmul *Rec* pentru a recupera secretul $\mathcal{S}$.

Schemele de partajare se clasifică in funcţie de cantitatea de informaţie secretă pe care o pot obţine persoanele care nu fac parte din $\mathcal{A}$ [7]:

- *Sisteme perfecte de partajare*: componentele nu oferă nici o informatie teoretică despre $\mathcal{S}$ indiferent de resursele computaţionale.
- *Sisteme statistic sigure*: o fracţiune de informaţie este dezvaluită despre $\mathcal{S}$ independent de puterea computională a adversarului.
- *Sisteme computaţional-sigure de partajare*: se bazează pe faptul ca reconstituirea lui $\mathcal{S}$ se reduce la o problema *dificilă* (spre exemplu problema Diffie-Hellman [8]) în lipsa unor informaţii oferite doar grupului de acces $\mathcal{A}$.

În continuare vom prezenta cateva sisteme perfecte de partajare utilizate în cadrul unor arhitecturi pentru stocarea fisierelor pe o durata indelungată.

2.1 Istoric

Primele scheme de partajare au fost dezvoltate independent de Shamir şi Blakley in 1979 [9, 10].

Denumite şi scheme majoritare (k, n), acestea rezolvau cazul în care oricare grup de participanţi cu un număr mai mare sau egal decât k (mărimea pragului) poate reconstitui secretul $\mathcal{S}$ din componentele primite de la dealer. Dacă schema este

perfect *sigură* atunci oricare grup cu un număr de participanţi mai mic decat k nu obţine vreo informaţie despre $\mathcal{S}$.

Schemele majoritare (spre exemplu schema Shamir) sunt insuficiente pentru a permite partajarea pentru anumite structuri de acces. Considerăm cazul în care vrem sa partajam un secret între 4 participanţi: P_1, P_2, P_3, P_4 astfel încât $\{P_1, P_2\}$ şi $\{P_3, P_4\}$ să fie singurele mulţimi autorizate pentru reconstrucţia secretului S (i.e. $\mathcal{A} = \{\{P_1, P_2\}, \{P_3, P_4\}\}$). În mod evident, problema nu poate fi rezolvată cu o structură de acces de tip prag: anumite mulţimi de 2 participanţi trebuie să poată reconstrui secretul ($\{P_1, P_2\}, \{P_3, P_4\}$), în timp ce altele nu ($\{P_1, P_3\}, \{P_1, P_4\}, \{P_2, P_3\}, \{P_2, P_4\}$)

Astfel de scheme de partajare pentru structuri de acces generale au fost dezvoltate de Ito, Saito şi Nishizeki, realizând o generalizare a schemei Shamir [11]. Benaloh şi Leichter au demonstrat ca schemele de partajare de tip prag nu pot fi folosite pe structuri general monotone (familie de submulţimi ale lui $\mathcal{P}$ cu proprietatea că dacă $A \in \mathcal{A}$ şi $A \subset A'$, atunci $A' \in \mathcal{A}$) şi obţin o construcţie mai eficientă ca Ito et. al din punct de vedere al numărului de componente distribuite participanţilor [12].

2.2 Schema unanimă

Presupunând ca vrem să impărţim un secret $\mathcal{S}$ la n participanţi astfel încât $\mathcal{S}$ sa poată fi recuperat doar daca toţi cei n participanţi îşi combină componentele pe care le deţin. Metoda este echivalentă cu o schemă (n, n) majoritară. Un exemplu este schema introdusă de Karin, Greene şi Hellman (Fig.1) [13].

2.3 Schema Shamir

Schema Shamir oferă mai multă flexibilitate decât schema unanima prin faptul ca oricare k (sau mai multi) participanţi din cei n pot recupera $\mathcal{S}$, însă mai puţin de k participanţi nu obţin nicio informaţie despre $\mathcal{S}$. Schema Shamir este deci o schemă (k, n) majoritară.

Intuitiv, având k puncte in plan (x_i, y_i), $x_i \neq x_j$ $i, j \in \{1, 2, \ldots, k\}$ $\forall i \neq j$, există o curbă polinomială unică care trece prin ele. În schimb, pentru a defini o curbă polinomială de grad k care trece prin $k-1$ puncte date, există o infinitate de soluţii. Evident, orice submulţime de valori s_i de mărime egală cu k este suficientă şi necesară pentru a reconstrui polinomul f. Dupa interpolarea componentelor deţinute de cel puţin k dintre participanţi, secretul $\mathcal{S}$ se determină ca fiind $f(0)$ (Fig. 2) [10].

Pentru un atacator care deţine chiar şi $k-1$ valori s_i, acesta nu determină nimic despre $\mathcal{S}$, spaţiul de soluţii posibile fiind identic faţă de situaţia în care nu reuşeşte sa obţină vreo componentă.

Iniţializare:

– Fie $S \in Z_q$ unde $q > 1$ şi q prim;
– Fie n numărul de participanţi;

Generare: Dealerul $\mathcal{D}$:

– Alege $n-1$ valori aleatoare $s_i \leftarrow^R Z_p$, $i \in \{1, 2, \ldots, n-1\}$;

– $s_n = S + \sum_{i=1}^{n-1} s_i \pmod q$;

Distribuţie: Dealerul $\mathcal{D}$:

– transmite în mod sigur participantului P_i componenta s_i, $i \in \{1, 2, \ldots, n\}$;

Reconstrucţie: Cei n participanţi:

– Calculează $S = \sum_{i=1}^{n} s_i \pmod q$.

Fig. 1: Schema unanimă [13]

2.4 Schema Ito, Saito şi Nishizeki

În continuare vom descrie modalitatea de distribuire a componentelor de la care au pornit Ito, Saito şi Nishizeki pentru ca schema sa aiba o structură de acces $\mathcal{A} \subseteq 2^P$ (submulţime a setului de participanţi) monotonă (i.e. $\forall A \in \mathcal{A}, A \subseteq A' \Rightarrow A' \in \mathcal{A}$) [11]. Folosind construcţia unei scheme majoritare (k, n) autorii au reuşit să descrie elementele din $\mathcal{A}$ folosind rezultatul unei reuniuni de mulţimi de componente cu un număr de elemente mai mare sau egal decat k (Fig. 3) [11]. Notaţia $x : Pr$, înseamnă că x are proprietatea Pr.

Dezvantajul acestei structuri este numărul de componente necesar pentru o structură de acces oarecare $\mathcal{A}$. Un mod simplu de construire al funcţiei $Assign$ este următorul: pentru fiecare mulţime minimală $A \in \mathcal{A}$ ($\forall B \in \mathcal{A}$, $B \neq A, B \not\subset A$) se foloseşte o schemă de partajare unanimă $(|A|, |A|)$ a lui $\mathcal{S}$ pentru participanţii din A.

Exemplul 1. Fie structura de acces $\mathcal{A} = \{\{P_1, P_2\}, \{P_1, P_3, P_4\}\}$.

– Generăm componentele s_1, s_2 a.î. $s_1 \oplus s_2 = \mathcal{S}$ cu ajutorul schemei unanime $(2, 2)$,

– Generăm componentele s_3, s_4, s_5 a.î. $s_3 \oplus s_4 \oplus s_5 = \mathcal{S}$ cu ajutorul schemei unanime $(3, 3)$

Participanţii primesc in felul următor componentele:

Iniţializare:

- Fie $S \in Z_q$ unde $q > 1$ şi q prim;
- Fie n numărul de participanţi a.i $q > n$;
- Fie k numărul minim de componente puse in comun pentru a determina pe $\mathcal{S}$;

Generare: Dealerul $\mathcal{D}$:

- Alege n valori distincte $x_i \leftarrow^R Z_q$, $i = 1, 2, \ldots, n$;
- Alege $a_i \leftarrow^R Z_q$, $i \in \{1, 2, \ldots, k-1\}$, $a_{k-1} \neq 0$;
- Construieşte polinomul $f(x) = a_{k-1}x^{k-1} + a_{k-2}x^{k-2} + \cdots + a_1 x + \mathcal{S}$;
- Calculează $s_i = f(x_i)$, $i \in \{1, 2, \ldots, n\}$;

Distribuţie: Dealerul $\mathcal{D}$:

- Transmite participantului P_i componenta s_i, $i \in \{1, \ldots, n-1\}$;

Reconstrucţie: Orice mulţime cu dimensiunea k (sau mai mare) de participanţi distincţi $P_1, P_2, \ldots, P_k$:

- Interpolează punctele s_i pentru a obţine polinomul f:

$$f(x) = \sum_{i=1}^{k} s_i \prod_{1 \leq j \leq k, j \neq i} \frac{x - x_j}{x_i - x_j} \tag{1}$$

- Află secretul reconstruit $S = f(0)$.

Fig. 2: Schema Shamir [10]

- $P_1 : \{s_1, s_3\}$;
- $P_2 : \{s_2\}$;
- $P_3 : \{s_3\}$;
- $P_4 : \{s_4\}$;

3 Sisteme de stocare de lunga durata

In acesta sectiune vom arata cateva intrebuintari ale schemelor de partajare. Consideram cazul in care vrem sa stocham rapoarte medicale, imagini, documente clasificate pe un timp indelungat intr-un mediu electronic. Pe parcursul timpului, pot apare in schimb, diverse probleme precum dezastre naturale, defectiunea unor componente hardware, eroare umana, etc [14]. Un sistem de stocare necesar nevoilor noastre trebuie sa satisfaca cel putin urmatoarele 3 conditii:

Iniţializare:

- Fie q un număr prim q, $q > 1$, $z \in \mathbb{N}$ nenul şi $\mathcal{C} = GF(p^z)$;
- Fie $S \in \mathcal{C}$ secretul;
- Fie structura de acces $\mathcal{A}$;
- Fie n numărul de participanţi;

Generare: Dealerul $\mathcal{D}$:

- Alege n valori distincte $x_i \leftarrow^R Z_q$, $i = 1, 2, \ldots, n$;
- Alege $a_i \leftarrow^R \mathcal{C} \setminus \{0\}$, $i \in \{1, 2, \ldots, k-1\}$, $a_{k-1} \neq 0$;
- Construieşte polinomul $f(x) = a_{k-1}x^{k-1} + a_{k-2}x^{k-2} + ... + a_1x + \mathcal{S}$;
- Atribuie $s_i = f(x_i)$ $i \in \{1, 2, \ldots, n\}$; Fie $Shares = \{s_1, \ldots, s_n\}$;
- Alege $D_i \subseteq Shares$ $1 \leq i \leq n$;
- Alege funcţia $Assign : P \rightarrow 2^Q$:
  - $Assign(P_i) = D_i$ $1 \leq i \leq n$
  - $\mathcal{A} = \left\{ Q \subseteq Shares : \left| \bigcup_{P_i \in Q} Assign(P_i) \right| \geq k \right\}$;

Distribuţie: Dealerul $\mathcal{D}$:

- Transmite participantului P_i componenta $Assign(P_i)$, $i \in 1, 2, \ldots, n$;

Reconstrucţie: Participanţii din structura de acces $\mathcal{A}$:

- Procedeaza identic ca in schema Shamir.

Fig. 3: Schema Ito, Saito, si Nishizeki [11]

- Disponibilitatea: Informatia trebuie sa ramana accesibila tot timpul, in ciuda erorilor de tip hardware.
- Integritatea: Abilitatea sistemului de a raspunde cererilor intr-un mod care garanteaza corectitudinea lor.
- Confidentialitatea: O persoana care nu face parte din grupul de acces sa nu obtina permisiunea de a afla informatii de orice fel despre datele existente in sistem.

3.1 Criptare VS scheme de partajare

Una dintre solutiile existente pentru a construi acest sistem ar putea fi criptarea datelor folosind o cheie inainte de inserarea lor in spatiul de stocare. In momentul in care un user autorizat doreste sa efectueze o citire a unor date, intrebuinteaza

cheia potrivita pentru a le decripta. In practica exista algoritmi de criptare eficienti precum AES insă aceaştia nu garantează confidenţialitatea datelor în cazul în care avem de a face cu un adversar fara o limita computationala. Un dezavantaj al criptarii este adminstrarea cheilor, standardele de securitatea schimbanduse in fiecare an. De fiecare data cand cheile sunt inlocuite atunci este necesara recriptarea datelor de pe fiecare baza de date. Cu cat disponibilitatea este mai mare - numarul de noduri duplicate creste- recriptarea lor devine o operatie costisitoare.

Majoritatea tehnicilor de criptarea se bazeaza pe dificultatea factorizarii unui numar sau cea a calcularii logaritmului discret insa o data cu posibila dezvoltare a calculatoarelor cuantice aceste probleme nu vor mai fi atat de dificile [15].

4 Sisteme de stocare de lungă durată bazate pe scheme de partajare

O alternativă la soluţia cu criptare care asigură atât confidenţialitate cât şi redundanţa necesară este intrebuinţarea sistemelor de stocare de lungă durată bazate pe scheme de partajare [14, 16, 17].

4.1 PASIS

PASIS este o soluţie pentru un sistem descentralizat care oferă beneficii precum securitate, redundanţă a datelor si auto-întreţinere [16] Structurile descentralizate împart informaţia la mai multe noduri folosind scheme de redundanţa precum "Redundant Array of Independent Disks" (RAID) pentru a asigura performanţa, scalabilitatea sistemului dar şi integritatea datelor [18]. RAID reprezintă o tehnologie ce combină 2 concepte ortogonale precum data striping (aranjarea datelor pe discuri multiple într-o manieră secvenţială - Fig **??**) şi redundanţă pentru o disponibilitate ridicată [19].

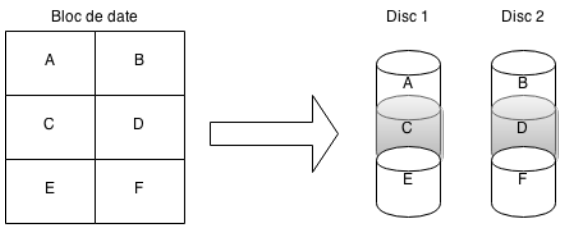

Fig. 4: Procesul de striping aplicat unui bloc de date

PASIS foloseşte schemele de partajare pentru a distribui informaţia nodurilor de stocare dintr-o reţea. Aceasta introduce agenţi pe partea clientului pentru a scrie sau şterge date din noduri, dar şi agenţi pentru mentenanţă. Componentele obţinute în urma partajării unui fisier sunt stocate in reţea cu ajutorul agenţilor (Fig. 5). Pe lângă conţinutul brut al componentelor se adaugă metadate pentru a reţine adresa nodului din reţea la care au fost stocate dar şi noua denumire cu care este salvată in reţea.

Considerând o schemă de partajare majoritară (k, n) unde oricare din cei k participanţi pot reconstitui fisierul, dar mai puţin de k nu obţin nicio informaţie dintr-un total de n componente.

Atunci când un participant iniţiază o cerere pentru a citi un fisier, agentul PASIS aflat local procedează dupa cum urmează:

- Caută numele celor n componente care alcătuiesc fişierul într-un serviciu care listează toate datele.
- Iniţiaza cereri de citire la cel puţin k din cele n noduri.
- În caz ca acesta nu primeşte cel puţin k răspunsuri se întoarce la pasul anterior încercând interogări la noduri diferite.
- Reconstituie fişierul obţinut din cele k componente.

Operaţia de scriere este similară cu cea de citire, aceasta oprindu-se atunci când în cel puţin $n - k + 1$ noduri s-au stocat cu succes componente. În articol se menţionează şi compromisul de spaţiu-timp folosit de PASIS în funcţie de alegerea schemei de partajare. Spre exemplu, cererile de citire pot fi făcute la acele k noduri pentru care răspunsurile recente au fost cele mai rapide. Autorii specifică soluţii pentru auto mentenanţa sistemului cu ajutorul resurselor umane prin monitorizarea periodică stării sistemului folosind log-uri sau ajustarea parametrilor din cadrul schemei de partajare.

4.2 GridSharing

În 2005, Subbiah şi Blough propun o nouă abordare pentru a construi un sistem de stocare securizat şi tolerant la erori numit GridSharing [17].

Schema Shamir nu oferă siguranţă în ceea ce priveşte detectarea sau actualizarea unor componente incorecte introduse de un atacator. Metoda cea mai des folosită este determinarea validităţii componentelor prin utilizarea semnăturilor electronice. Aceasta este realizată prin scheme de verificare non-interactive precum cea a lui Feldman construită pe baza schemei Shamir [20] (Fig. 6)

Pentru verificarea componentei $s_i = f(i)$, participantul i probează ecuaţia:

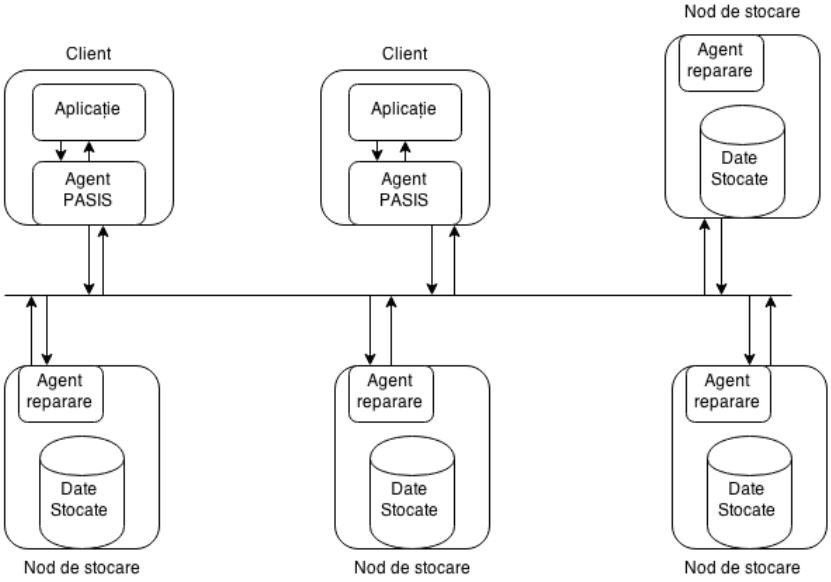

Fig. 5: Arhitectura PASIS cu 4 noduri şi 2 clienţi [16].

Iniţializare: Folosind notaţiile din Fig. 2, considerăm polinomul f generat în urma Schemei Shamir.

– Se alege un $g \in Z_q$ generator al lui Z_q;

Generare: Dealerul $\mathcal{D}$:

– Calculează *angajamentele*: $c_0 = g^S$, $c_j = g^{a_j}$, $j \in \{1, \ldots, k\}$;

Distribuţie: Dealerul $\mathcal{D}$:

– Distribuie *angajamentele* c_j, $j \in \{0, \ldots, k\}$ fiecărui participant $i \in \{1, \ldots, n\}$;

Fig. 6: Schema Feldman [20]

$$g^{s_i} = c_0 c_1^i c_2^{i^2} \ldots c_t^{i^t} = \prod_{j=0}^{k} c_j^{i^j} = \prod_{j=0}^{k} g^{a_j i^j} = g^{\sum_{j=0}^{k} a_j i^j} = g^{f(i)} \tag{2}$$

Subbiah şi Blough folosesc un sistem care înlocuieşte schemele de verificare cu o schemă de partajare unanimă XOR (considerăm cazul $q = 2$ în Fig. 1) pentru a păstra securitatea construcţiei. În cazul detectării componentelor incorecte, este adoptată o strategie de tipul replicate-and-voting. Componentele sunt replicate pe un număr mare de servere astfel încât determinarea validităţii va fi stabilită în funcţie de numărul de servere care le conţin.

Se identifică 3 tipuri de defecţiuni care pot apărea pe serverele unde sunt stocate datele:

- Abandonări: un server este *abandonat* dacă nu mai raspunde vreunui mesaj din reţea şi s-a oprit din a mai efectua vreo operaţie.

- Bizantine: atunci când serverul nu respectă întotdeauna protocoalele iniţiale iar componentele salvate local au fost compromise.

- Scurgeri de informaţii: serverul execută protocoalele corect dar e posibil ca un adversar să fi obţinut componentele stocate.

Primele 2 modele definite mai sus sunt preluate din calculul cu sisteme distribuite. Cel de-al 3-lea model a fost introdus pentru a defini atacatorul care foloseşte vulnerabilităţile cu intenţia de a *învaţa* din informaţii.

Arhitectura GridSharing constă in N servere unde cel mult c servere pot fi abandonate, b servere bizantine şi l cu scurgeri de informaţii. Cele N pot fi aranjate într-un grid cu r linii şi N/r coloane (considerăm pentru simplitate că $N \pmod r = 0$). Caracteristicile modelului bizantin şi cel specific scurgerilor de informaţii permit dezvăluirea componentelor unui adversar de pe cel mult $l + b$ servere.

Exemplul 2. Notăm $\binom{x}{y}$ fiind combinări de x elemente grupate câte y. Considerăm ca împarţim un secret $\mathcal{S}$ la 4 linii (participanţi) astfel încât sistemul să permită 2 componente de tip b, 1 componentă de tip l şi 20 servere. În cazul acesta vom folosi o schemă majoritară XOR $\left(\binom{4}{3}, \binom{4}{3}\right) = (4, 4)$.

Vom avea 4 componente, (s_1, s_2, s_3, s_4) a.î. $s_1 \oplus s_2 \oplus s_3 \oplus s_4 = \mathcal{S}$. Distribuirea se face in felul următor:

- Serverele situate pe prima linie primesc s_1

- Serverele situate pe a 2-a linie primesc s_2

- Serverele situate pe a 3-a linie primesc s_3

- Serverele situate pe a 4-a linie primesc s_4

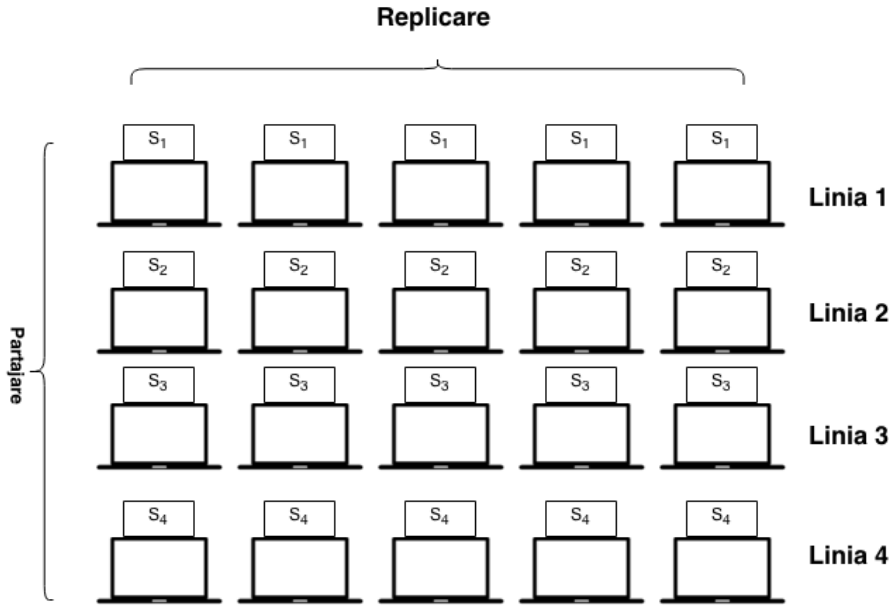

Fig. 7: GridSharing cu 4 linii, 20 servere dintre care 2 bizantine, 1 cu scurgeri de informaţii [17].

4.3 POTSHARDS

În 2007 este propus un nou sistem care combină caracteristicile PASIS şi GridSharing adăugând posiblitatea de migrarea a datelor la noduri noi: Protection Over Time, Securely Harboring And Reliably Distributing Stuff (POTSHARDS) [14].

POTSHARDS poate fi gândit ca o aplicaţie pe partea clientului care comunică cu o mulţime de noduri (arhive) independente, folosind reconstrucţia componentelor într-un mod securizat şi semnături algebrice pentru a asigura un grad ridicat de păstrare a integritaţii fişierelor [21].

Confidenţialitatea este asigurată prin adoptarea unei scheme de partajare XOR unanimă, la fel ca în GridSharing.

Ca prim pas, POTSHARDS preprocesează fişierul într-un obiect, partajează obiectul în fragmente la care adaugă meta-date, numite *shards* (Fig. 8) [14]. Acestea sunt trimise apoi arhivelor independente, fiecare având propriul domeniu de securitate, localizate in *regiuni*. Pentru a reconstitui cu succes informaţia iniţială, meta-datele shard-urilor conţin detalii despre structura pointerilor aproximativi,

indicând regiunea în care se află următorul shard. Pointerii aproximativi sunt folositi pentru a reconstitui intreaga arhiva doar din shard-uri.

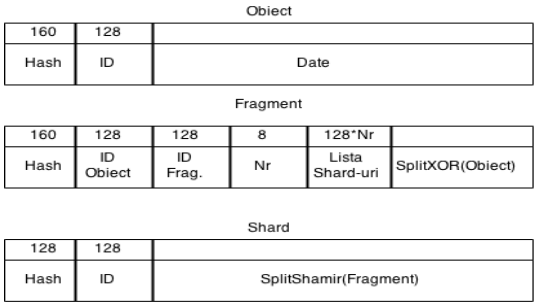

Fig. 8: Entităţi de date in POTSHARDS. Nr e numărul de shard-uri produse de un fragment. $SplitXOR$ reprezintă o componentă rezultată în urma partajarii unanime XOR. Analog $SplitShamir$ reprezintă o componentă rezultată în urma partăjarii folosind schema Shamir. [14]

Procesul de fragmentare a datelor este prezentat in Fig. 9.

Pentru ca reconstituirea unui fişier sa fie fezabilă unui utilizator, acestuia îi este întoarsă o listă cu locaţiile exacte shard-urilor corespunzătoare. Obţinerea unui shard de către un atacator nu este folositoare, pentru a detecta următorul shard, un atac brut force constă în cereri multiple în zona indicată de pointerul aproximativ. Un astfel de atac nu va trece neobservat de POTSHARDS deorece unul dintre scopurile sale este să stocheze datele într-un mod cat mai uniform distribuit. [14]

5 Sistemul de stocare Alouneh et al.

Alouneh et al. propun un sistem pentru stocarea sigură a datelor un timp indelungat folosind schema Shamir cu câteva modificări. Aceste schimbări se vor arăta esenţiale mai târziu în menţinerea securităţii [22].

Pentru stocarea unui fişier în sistem (abordând filozofia majorităţii sistemelor de operare - orice este un fişier), o aplicaţie client îl împarte în blocuri de octeţi de lungime k. Pentru fiecare bloc, octeţii devin coeficienţii unui polinom f de grad k, componenta corespunzătoare participantului i va fi reprezentată de valoarea lui $f(i)$, $i = \{1, 2, \ldots, n\}$. Menţionăm că toate operaţiile se vor efectua in $GF(256)$ modulo un polinom ireductibil (în implementarea sistemului, autorii folosesc $x^8 + x^5 + x^3 + x + 1$). Procedeul este descris in detaliu in Fig 10.

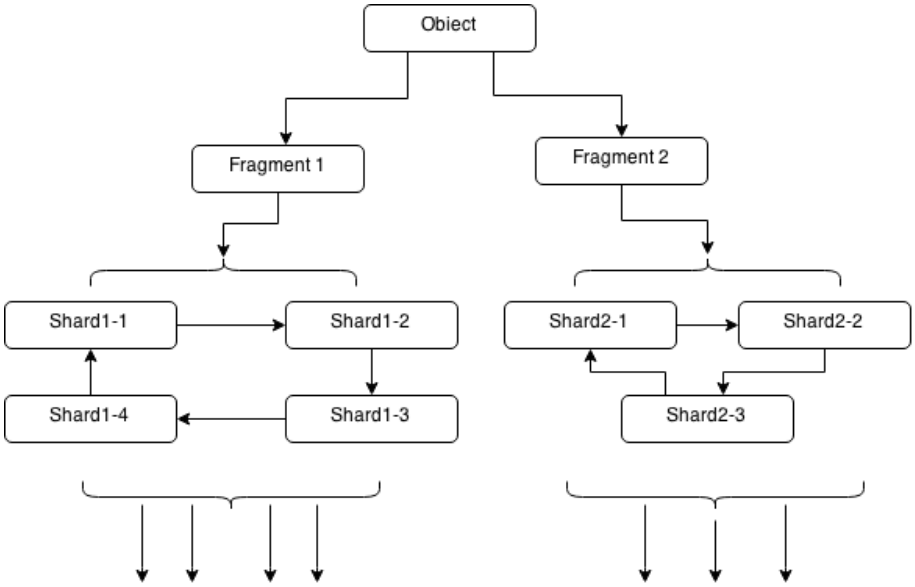

Fig. 9: Distribuirea unui obiect in POTSHARDS

Date de intrare: Un fişier binar $\mathcal{S}$;

Date de ieşire: n fişiere binare distribuite la noduri din reţea;

Procesarea componentelor: Aplicaţia existentă pe partea clientului:

– Dacă $\mathcal{S}$ nu are o lungime divizibilă cu k:
  • Concatenează la sfârsitul lui $\mathcal{S}$ octeţi până când $len(\mathcal{S}) \pmod k = 0$;
– Împarte $\mathcal{S}$ în blocuri de lungime k;
– Repetă pentru fiecare bloc B_t de lungime k:
  • Construieşte polinomul $f(x) = B_{t_{k-1}}x^{k-1} + B_{t_{k-2}}x^{k-2} + ... + B_{t_1}x + B_{t_0}$;
  • Calculează $f(i)$ pentru $1 \leq i \leq n$;

Distribuţie: Aplicaţia la nivelul clientului:

– Distribuie componenta $f(i)$ nodului din reţea cu indicele i:

Fig. 10: Schema Alouneh et al. - Generare [22]

Exemplul 3. Vom exemplifica modul de calcul în $GF(256) \pmod{g(x)}$ unde $g(x) = x^8 + x^4 + x^3 + 1$. Luăm polinomul $f(x) = 10 + 15x$ corespunzător unui

fişier format din octeţii (în această ordine) 10 15.

$$\begin{aligned} f(01) \pmod{g(x)} &= 10 + 15 \pmod{g(x)} \\ &= (x^4) + (x^4 + x^2 + 1) \pmod{g(x)} \\ &= x^2 + 1 = 000000101_{(2)} \\ &= 05_{(16)} \end{aligned} \tag{3}$$

$$\begin{aligned} f(02) \pmod{g(x)} &= 10 + 15 \cdot 02 \pmod{g(x)} \\ &= (x^4) + (x^5 + x^3 + x) \pmod{g(x)} \\ &= 00111010_{(2)} \\ &= 3A_{(16)} \end{aligned} \tag{4}$$

Reconstituirea unui fişier (Fig. 11) se realizează din orice mulţime de componente A de dimensiune minim k folosind interpolarea Lagrange (utilizată şi în cadrul schemei Shamir):

$$f(x) = \sum_{i \in A} f(i) \prod_{j \in A, j \neq i} \frac{x - j}{i - j} \tag{5}$$

Exemplul 4. Vom exemplifica interpolarea conform ecuaţiei 5 folosind componentele (3) şi (4) calculate în exemplul anterior:

$$\begin{aligned} f(x) &= 05(x - 02)(01 - 02)^{-1} + 3A(x - 01)(02 - 01)^{-1} \\ &= 05(x - 02)03^{-1} + 3A(x - 01)03^{-1} \\ &= F6(05 + 3A)x + F6(05 \cdot 02 + 3A \cdot 01) \\ &= F6 \cdot 3F \cdot x + F6 \cdot 30 = 15x + 10 \end{aligned} \tag{6}$$

Noutatea arhitecturii constă in diminuarea redundanţei componentelor la un factor de k, spre deosebire de sistemele descrise in Secţiunile 4.1, 4.2 şi 4.3. Reducerea spaţiului ocupat de componente este datorat înlocuirii coeficienţilor generaţi aleator din schema Shamir cu octeţii din fişierul ce va fi partajat, fiecare componentă având nevoie de $1/k$ din dimensiunea fişierului $\mathcal{S}$.

Abordarea acestei metode deterministe conduce la insecuritatea sistemului, Alouneh et al. afirmând în mod eronat că securitatea este indusă în mod automat de schema Shamir.

6 Rezultate obţinute

Împreună cu mentorul am analizat sistemul Alouneh et al. prezentat în Secţiunea 5 [22]. Am indentificat erori majore ale acestuia şi am implementat sistemul descris de autori pentru a demonstra practic, nu doar teoretic anumite greşeli pe care le vom evidenţia în următoarele secţiuni.

Date de intrare: Cel puţin k componente provenite din noduri (distincte);

Date de ieşire: Fişierul binar original $\mathcal{S}$;

Reconstrucţie: Aplicaţia existenta pe partea clientului:

- Repetă pentru fiecare bloc al lui $\mathcal{S}$:
  - Calculează prin interpolare coeficienţii lui $f(x) = B_{t_{k-1}}x^{k-1} + B_{t_{k-2}}x^{k-2} + ... + B_{t_1} + B_{t_0}$
  - Reconstituie blocul B_t
- Şterge octeţii de la sfârşitul fişierului adăugaţi la generare.

Fig. 11: Schema Alouneh et al. - Reconstrucţie [22]

6.1 Vulnerabilităţi evidenţiate

Spre deosebire de schema Shamir, unde coeficienţii (cu excepţia termenului liber, care este egal cu secretul) sunt aleşi într-un mod aleator uniform, sistemul propus de Alouneh et al. foloseşte o schemă modificată Shamir în care coeficienţii sunt extraşi din conţinutul fişierelor originale. Alegerea este motivată de faptul ca mulţimea componentelor şi efortul computaţional depus pentru generarea coeficienţilor se reduce la un factor de k ori, spre deosebire de schema Shamir.

Datorită determinismului, partajarea unui fişier de mai multe ori implică obţinerea de componente identice. Determinismul duce la câteva atacuri simple în momentul în care un atacator obţine informaţiile stocate într-un nod, indiferent de mărimea pragului folosit în metoda de partajare. În acest sens, am evidenţiat 2 atacuri simple în cazul în care componentele sunt calculate in ordine $(f(01), f(02), \dots)$:

- Detectarea tipului unui fişier
- Detectarea conţinutului unui fişier

Deoarece Alouneh et al. nu menţioneaă o metoda de padding, am indicat că un atac bazat pe felul în care se realizează completarea fişierului (padding) $\mathcal{S}$ înainte de partajarea sa poate fi fezabil, în condiţiile în care s-a demonstrat ca această alegere este esenţială în păstrarea securităţii [23].

6.1.1 Detectarea tipului de fişier

În tehnologia informaţiei, la inceputul fiecărui fişier se află o secvenţă de octeţi (denumită semnatură sau antet) cu rolul de a identifica tipul acestuia. Tabelul 1 indică 7 din cele mai uzuale antete.

Se consideră cazul în care partajarea unui fişier *pdf* se face cu ajutorul sistemului descris in Secţiunea 5, folosind $k \leq 4$. Polinomul corespunzător $f(x)$ va fi intotdeauna acelaşi. Presupunând ca numerotarea nodului i este aceeaşi, putem determina cu uşurinţă dacă este stocat un fişier *pdf* fără a lua în calcul conţinutul fişierului. Acest atac este fezabil deorece valoarea lui k este publică iar i poate sa fie descoperit în momentul distribuirii.

Cu alte cuvinte, dacă un adversar obţine controlul unui singur nod, bazându-se doar pe valoarea primei componente poate detecta tipul unui fişier.

Pentru a exemplifica, un adversar poate distinge cu probabilitate ridicată între fişierele *doc, gif, pdf, png, rar, wav* şi *zip*. În Tabelul 2 avem generate componentele pentru $k = 2$ şi $n = 5$. Dacă un adversar descoperă valoarea primului nod iar prima componentă este 14 atunci acesta ştie cu certitunde că aceasta corespunde un fişier *gif*. Dacă obţine accesul nodului 4 şi citeşte valoarea 205 atunci ştie că fişierul este de tipul *rar*. Daca citeşte valoarea 27 de pe primul nod atunci ştie că poate fi un *wav* sau *zip*; dar poate să distingă cele 2 fişiere cu probabilitate 1 dacă dezvăluie o singură valoare de pe celelalte noduri (2, 3, 4 sau 5) pentru că valorile sunt distincte.

Tabel 1: Semnături de fişiere

Tip de fişier	Primii 4 octeţi
doc	D0 CF 11 E0
gif	47 49 46 38
pdf	25 50 44 46
png	89 50 4E 47
rar	52 61 72 21
wav	52 49 46 46
zip	50 4B 03 04

Tabel 2: Componentele primului bloc ($k = 2$)

Tip fisier	Nod 1 ($i = 1$)	Nod 2 ($i = 2$)	Nod 3 ($i = 3$)	Nod 4 ($i = 4$)	Nod 5 ($i = 5$)
doc	31	85	154	193	14
gif	14	213	156	120	49
pdf	117	133	213	126	46
png	217	41	121	210	130
rar	51	144	241	205	172
wav	27	192	137	109	36
zip	27	198	141	103	44

6.1.2 Detectarea de conţinut

Multe documente urmează un anumit tipar precum contracte, chitanţe, bonuri fiscale sau curriculum vitae. Deoarece majoritatea conţinutului rămâne neschimbat, există o probabilitate destul de mare ca multe componente sa aiba aceeaşi valoare. O dată ce un adversar reuşeşte să determine componentele unui nod, poate determina prin analogie tipul de conţinut al fişierului original.

Fişierele vulnerabile sunt cele care conţin o secvenţă de octeţi periodică (imagini cu un pattern repetitiv) sau cele care au preponderent octeţi nuli (valoarea componentelor asociată majorităţii blocurilor fiind 0). Spre deosebire de prima metodă, aceasta nu necesită compararea cu un al 2-lea fişier, tratând componentele de sine stătător. Multiple componente identice indică existenţa unor secvenţe repetitive în conţinutul fişierului.

6.2 Implementare şi rezultate practice

Pentru a arăta aplicabilitatea rezultatelor în practică, am implementat propunerea descrisa in Secţiunea 5 şi am testat pe câteva cazuri.

În cadrul implementării am folosit limbajul Python 3.0 sub sistemul de operare ArchLinux. Python este un limbaj high-level, permiţând programatorilor să exprime concepte în mai puţine linii de cod faţă de C++ sau Java şi este disponibil sub licenţă open-source [24]. Pentru a realiza comunicarea între procese am folosit Cerealizer iar distribuţia componentelor a fost generată grafic cu ajutorul pachetului Matplotlib [25,26]. De asemenea am folosit sistemul de versionare Git iar în prezent codul folosit este găzduit de GitHub [27, 28].

Având în vedere că articolul original nu menţionează o metodă de padding, am considerat o metodă standard pentru a completa octeţii ultimului bloc: alipim la sfârşitul lui $\mathcal{S}$ octeţii 80 00 ... 00 00 până când lungimea ultimului bloc ajunge la k octeţi.

Menţionăm această metoda doar pentru completitudine, aceasta neafectând rezultatele, care consideră doar secvenţele de octeţi din antet sau de la începutul fişierului.

6.2.1 Detectarea tipului de fişier

Extindem analiza făcută în Secţiunea 6.1.1 asupra tipurilor de fişiere din Tabelul 1 pentru $k = 2$ şi creştem valoarea indicelui i până când 2 componente devin egale. Fie f_l polinomul de gradul 1 asociat primului bloc al fişierului aflat pe linia l. Analog f_c polinomul de gradul 1 asociat primului bloc al fişierului aflat pe coloana c.

Tabelul 3 prezintă valoarea maximă a nodul i pentru care $f_l(i) \neq f_c(i)$. Valoarea -1 indică lipsă de coliziuni ale lui $f_c(x)$, $f_l(x)$ pentru i, $i = 1, 2, \ldots, 255$ ($\nexists 1 \leq i \leq 255$ a.î. $f_l(i) = f_c(i)$)).

Deoarece pe diagonala principală toate componentele sunt identice pentru $k \leq 4$, poate fi ignorată. În Tabelul 3 observăm valoarea 0 pentru perechea (wav, zip) pentru că $f_{wav}(1) = f_{zip}(1) = 27$ (Tabelul 1).

Tabelele 4 şi 5 prezintă rezultatele pentru $k = 3$ si $k = 4$. Pentru $k \geq 5$ este nevoie de un antet cu mai mult de 4 octeţi.

Considerăm metoda de distribuire descrisă exact ca în Tabelul 10, şi anume nodul de stocare i primeşte componenta $f(i)$. Dacă un atacator preia controlul nodului cu valoarea i acesta poate face distincţia între tipul a 2 fişiere partajate cu probabilitate 1 dacă i e mai mic decât valoarea afişată în tabel.

Spre exemplu, un adversar care obţine accesul unui singur nod de stocare, iar partajarea a fost făcută pentru $k = 2$ poate distinge cu probabilitate 1 între un fişier *doc* sau *pdf* dacă $i \leq 194$. Deoarece $n > 194$ nu se întâmplă în practică, acest atac funcţionează. Faptul că multe valori sunt ridicate şi majoritatea sunt -1 (Tabelul 4), indică un nivel scăzut de securitate al schemei. În cazul celor cu -1 un adversar va caştiga întotdeauna cu probabilitatea 1, indiferent de indicele nodului de stocare pe care il obţine.

Precizăm că acest atac funcţionează doar dacă nodurile de stocare îşi păstrează indexul în cazul partajării repetate a.î. aplicaţia disponibilă pe partea de client calculează cele n valori $f(i_1), \ldots, f(i_2)$ pentru $i_1, \ldots, i_n$ distincte şi păstrează nodul j asociat lui i_j.

Tabel 3: Indicele maxim i a.î. componentele primului bloc sa fie distincte($k = 2$)

Tip Fişier	doc	gif	pdf	png	rar	wav	zip
doc	-	169	194	209	170	206	110
gif	169	-	133	137	75	-1	133
pdf	194	133	-	-1	115	151	133
png	209	137	-1	-	229	147	195
rar	170	75	115	229	-	-1	42
wav	206	-1	151	147	-1	-	0
zip	110	133	133	195	42	0	-

Tabel 4: Indicele maxim i a.î. componentele primului bloc sa fie distincte($k = 3$)

Tip Fişier	doc	gif	pdf	png	rar	wav	zip
doc	-	63	-1	-1	-1	-1	-1
gif	63	-	-1	-1	-1	-1	-1
pdf	-1	-1	-	164	-1	119	-1
png	-1	-1	164	-	143	122	129
rar	-1	-1	-1	143	-	143	-1
wav	-1	-1	119	122	143	-	172
zip	-1	-1	-1	129	-1	172	-

Tabel 5: Indicele maxim i a.î. componentele primului bloc sa fie distincte($k = 4$)

Tip Fişier	doc	gif	pdf	png	rar	wav	zip
doc	-	-1	38	95	1	95	98
gif	-1	-	-1	-1	167	-1	-1
pdf	38	-1	-	12	11	119	70
png	95	-1	12	-	243	95	148
rar	1	167	11	243	-	-1	94
wav	95	-1	119	95	-1	-	-1
zip	98	-1	70	148	94	-1	-

6.2.2 Detectarea de conţinut

Considerăm scenariul pentru detectarea conţinutului unui fişier, demonstrând practic cum un adversar poate să facă diferenţa între 2 componente stocate pe acelaşi nod aparţin unor documente similare. Atacul presupune accesul la un singur nod, indiferent de mărimea pragului k.

Pentru experiment am ales 3 fişiere *pdf* disponibile online la [29]:

- Europass Curriculum Vitae - BG, Bulgaria;
- Europass Curriculum Vitae - DK, Dannemark;
- Europass Mobility - RO, Romania.

Primele două fişiere reprezintă şablonul pentru un CV european în limba bulgară, respectiv daneză. Observăm că nu doar conţinutul şabloanelor diferă, dar şi limba în care au fost traduse. Cel de-al treilea fişier este complet diferit de primele două, fiind un document personal în limba română, folosit pentru înregistrarea cunoştiintelor dobândite într-o ţară europeană.

Pentru a arăta vulnerabilitatea sistemului Alouneh et al. [22], partajăm cele 3 fişiere folosind schema $(2, 4)$. Experimentul reprezintă o implementare practică a metodei prezentate in Fig. 10.

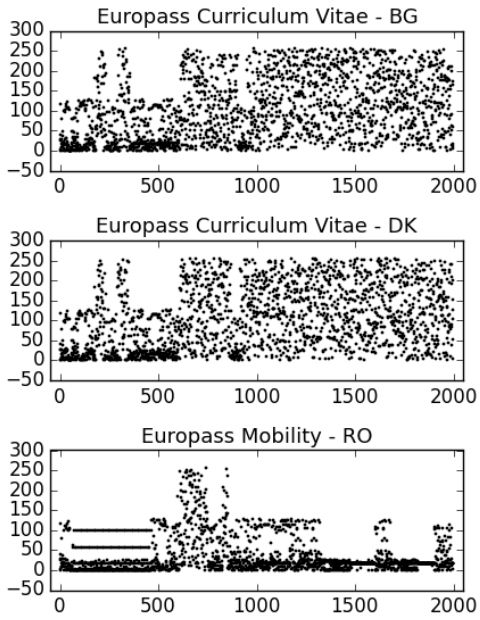

Fig. 12: Nod 1: Graficul componentelor pentru primele 2000 de blocuri

Fig. 12, 13, 14, 15 conţin 2000 de componente generate pentru cele 3 fişiere, stocate în baza de date corespunzătoare nodurilor $1, 2, 3$ şi 4. Presupunem că indicele i este fixat, polinoamele fiind evaluate în acelaşi nod i. Un adversar care obţine componentele unui nod poate deduce cu uşurinta graficul lor.

Considerăm că un adversar reuşeşte să obţină componentele din Fig. 12. Poate deduce uşor că primele două fişiere au un număr mare de componente identice iar cel de-al 3-lea este diferit de primele.

Din cauza şablonului pe care îl respectă cele două CV-uri, un adversar are nevoie în cazul de faţă doar de primele 500 de componente pentru a constata similiaritatea celor 2 cu o probabilitate ridicată. Diferenţa vizuala este evidentă între oricare din cele 4 grafice, deci un adversar poate reuşi să afle informaţii despre fişierele partajate indiferent de nodul de stocare vulnerabil.

Considerăm în continuare un alt scenariu, în care partajăm o imagine pentru care informaţia se repetă. Fie imaginea din Fig. 16 partajată folosind schema

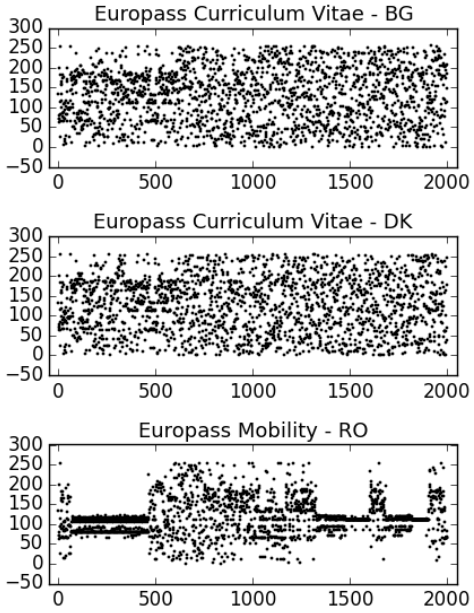

Fig. 13: Nod 2: Graficul componentelor pentru primele 2000 de blocuri

din Secţiunea 5 având 4 noduri iar informaţia stocată pe cel puţin 2 din ele pot reconstitui imaginea partajată iniţial. 10. La fel ca în exemplul precedent, Fig. 17, 18, 19, 20 reprezintă grafic componente imaginii din Fig. 16. Un adversar constată cu uşurinţă prin obţinerea informaţiilor stocate aflate pe un singur nod (indiferent de mărimea pragului k) că fişierul original conţine un pattern care se repetă.

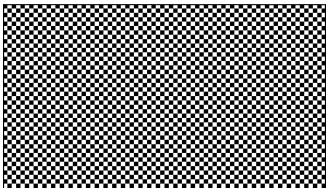

Fig. 16: Imagine conţinând pattern-uri

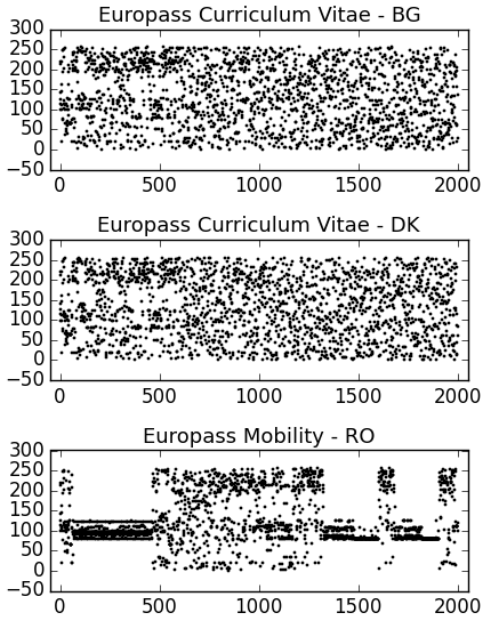

Fig. 14: Nod 3: Graficul componentelor pentru primele 2000 de blocuri

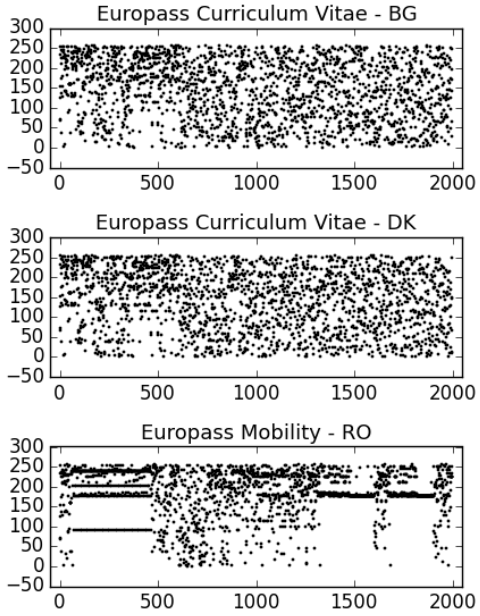

Fig. 15: Nod 4: Graficul componentelor pentru primele 2000 de blocuri

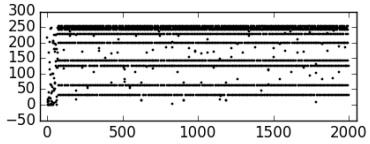

Fig. 17: Nod 1: Graficul componentelor pentru primele 2000 de blocuri

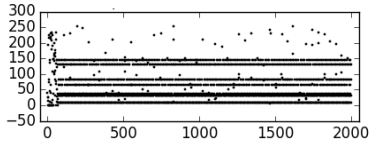

Fig. 18: Nod 2: Graficul componentelor pentru primele 2000 de blocuri

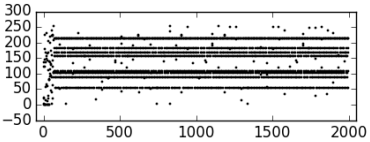

Fig. 19: Nod 3: Graficul componentelor pentru primele 2000 de blocuri

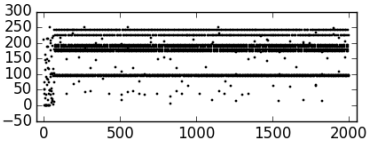

Fig. 20: Nod 4: Graficul componentelor pentru primele 2000 de blocuri

6.3 Publicarea rezultatelor

Pe baza rezultatelor obţinute am redactat (împreună cu mentorul) un articol, aflat momentan în procesul de recenzie la Journal of Control Engineering and Applied Informatics (jurnal indexat ISI,categoria C). [30]

Referinţe

1. WebSite: The Caesar cipher (2015) Ultima accesare: Februarie, 2015, http://www.cs.trincoll.edu/ crypto/historical/caesar.html.
2. WebSite: Hacking the Vigenere cipher (2015) Ultima accesare: Februarie, 2015, http://inventwithpython.com/hacking/chapter21.html.
3. Katz, J., Lindell, Y.: Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series). Chapman & Hall/CRC (2007)
4. Beimel, A.: Secret-sharing schemes: a survey. In: Coding and cryptology. Springer (2011) 11–46
5. Hillery, M., Bužek, V., Berthiaume, A.: Quantum secret sharing. Physical Review A **59**(3) (1999) 1829
6. Csirmaz, L.: The size of a share must be large. Journal of cryptology **10.4** (1997) 223–231
7. Martin, K.M.: Challenging the adversary model in secret sharing schemes. Coding and Cryptography II, Proceedings of the Royal Flemish Academy of Belgium for Science and the Arts (2008) 45–63
8. Boneh, D.: The decision Diffie-Hellman problem. In: Algorithmic number theory. Springer (1998) 48–63
9. Blakley, G.: Safeguarding cryptographic keys. Proceedings of the 1979 AFIPS National Computer Conference (1979) 313–317
10. Shamir, A.: How to share a secret. Commun. ACM **22**(11) (1979) 612–613
11. Ito, M., Saito, A., Nishizeki, T.: Secret sharing scheme realizing general access structure. Electronics and Communications in Japan (Part III: Fundamental Electronic Science) **72**(9) (1989) 56–64
12. Benaloh, J., Leichter, J.: Generalized secret sharing and monotone functions. In: Proceedings on Advances in Cryptology. CRYPTO '88, New York, NY, USA, Springer-Verlag New York, Inc. (1990) 27–35
13. Karnin, E.D., Member, S., Greene, J.W., Member, S., Hellman, M.E.: On secret sharing systems. IEEE Transactions on Information Theory **29** (1983) 35–41
14. Storer, M.W., Greenan, K.M., Miller, E.L., Voruganti, K.: Potshards - a secure, recoverable, long-term archival storage system. TOS **5**(2) (2009)
15. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on, IEEE (1994) 124–134
16. Wylie, J.J., Bigrigg, M.W., Strunk, J.D., Ganger, G.R., Kiliççöte, H., Khosla, P.K.: Survivable information storage systems. Computer **33**(8) (2000) 61–68
17. Subbiah, A., Blough, D.M.: An approach for fault tolerant and secure data storage in collaborative work environments. In: StorageSS. (2005) 84–93
18. Patterson, D.A., Gibson, G., Katz, R.H.: A case for redundant arrays of inexpensive disks (raid). SIGMOD Rec. **17**(3) (June 1988) 109–116
19. Chen, P.M., Lee, E.K., Gibson, G.A., Katz, R.H., Patterson, D.A.: Raid: High-performance, reliable secondary storage. ACM Comput. Surv. **26**(2) (June 1994) 145–185
20. Feldman, P.: A practical scheme for non-interactive verifiable secret sharing. In: Proceedings of the 28th Annual Symposium on Foundations of Computer Science. SFCS '87, Washington, DC, USA, IEEE Computer Society (1987) 427–438
21. Schwarz, T.J.E., Miller, E.L.: Store, forget, and check: Using algebraic signatures to check remotely administered storage. In: 26th IEEE International Conference on Distributed Computing Systems (ICDCS 2006), 4-7 July 2006, Lisboa, Portugal. (2006) 12

22. Alouneh, S., Abed, S., Mohd, B.J., Kharbutli, M.: An efficient backup technique for database systems based on threshold sharing. JCP **8**(11) (2013) 2980–2989
23. Vaudenay, S.: Security flaws induced by cbc padding - applications to ssl, ipsec, wtls. In: Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques: Advances in Cryptology. EUROCRYPT '02, London, UK, UK, Springer-Verlag (2002) 534–546
24. WebSite: Python Programming Language - official website (2015) Ultima accesare: Februarie, 2015, https://www.python.org/.
25. Hunter, J.D.: Matplotlib: A 2D graphics environment. Computing In Science & Engineering **9**(3) (2007) 90–95
26. WebSite: Cerealizer package (2015) Ultima accesare: Februarie, 2015, https://pypi.python.org/pypi/Cerealizer.
27. WebSite: Github - Official Website (2015) Ultima accesare: Februarie, 2015, https://www.github.com.
28. WebSite: Cod Github - Official Website (2015) Ultima accesare: Februarie, 2015, https://www.github.com/rdragos/splitting-scheme.
29. WebSite: Europass - download examples (2014) Ultima accesare: November, 2014, http://www.europass.cedefop.europa.eu/ro/home.
30. WebSite: Journal of Control Engineering and Applied Informatics (2015) Ultima accesare: Februarie, 2015, http://www.ceai.srait.ro/index.php/ceai/index.