UNIVERSITATEA BUCUREȘTI
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

Protocoale Criptografice de Grup

Lucrare de Licență

Coordonator:

Lect.dr. Ruxandra F. Olimid

Student:

Dragoș Alin Rotaru

BUCUREȘTI
2015

Abstract

Criptografia modernă se găsește pretutindeni în viața de zi cu zi de la ATM-uri, cartele telefonice la protocoale de autentificare, licitații electronice sau bani digitali, luând amploare o dată cu apariția sistemelor cu cheie publică. O definiție potrivită este *studiul științific al tehnicilor pentru a securiza informația digitală, tranzacțiile și calculul distribuit* [1]. În cadrul criptografiei există primitive multiple care stau la baza tutoror protocoalelor precum: semnături digitale, vot electronic, scheme de partajare, schimbul sigur de chei. Un rol important îl deține criptografia de grup, care permite definirea și analiza securității unor protocoale la care iau parte mai mult de 2 participanți (vot elecronic, semnături de grup, partajarea de secrete).

Păstrarea confidențialității și redundanței datelor pe perioade lungi de timp reprezintă o sarcină dificilă din cauza multiplelor probleme care pot apărea precum atacuri, erori umane, defecțiunea unor componente hardware sau dezastre naturale [2]. Problemele sunt cu atât mai actuale astăzi, când organizațiile își păstrează datele în cloud pentru scăderea costurilor necesare întreținerii unui centru propriu de date (datacenter), deci informațiile sunt stocate și procesate în afara unui control fizic al roprietarului.

Lucrarea de față analizează câteva dintre primitivele criptografice prezentând protocoale construite pe baza lor precum şi contribuţii personale în dezvoltarea lor [3, 4]. În final introducem o arhitectură implementată cu ajutorul schemelor de partajare, eliminând astfel câteva vulnerabilități prezente la sistemele de stocare descrise anterior.

Cuprins

Introducere şi organizarea lucrării

- TODO Completeaza descrierea
- Secțiunea **??** oferă o scurtă introducere în domeniul schemelor de partajare și descrie în detaliu construcțiile care vor fi folosite în secțiunile următoare.
- Secțiunea 4.3 conține cerințele pe care trebuie să le îndeplinească un sistem de stocare de lunga durată și compară sistemele de stocare de lungă durată care utilizează criptarea cu sistemele de stocare bazate pe scheme de partajare a secretelor. De asemenea, sunt descrise câteva sisteme considerate relevante pentru întelegerea contribuțiilor până în prezent.
- Secțiunea 4.4 introduce un sistem de stocare criptanalizat.
- Secțiunea 4.5 conține contribuțiile de cercetare, punând în evidență problemele cauzate de determinismul sistemului introdus în Secțiunea 4.4.

0.1 Introducere + Motivatie - deocamdata text reciclabil

Termenul de criptografie este definit in dicționarul Oxford ca fiind *arta de a scrie și a rezolva coduri*. Criptografia modernă s-a desprins de cea clasica în jurul anilor '80, motivând implementarea rigurozității matematice pentru

definirea construcțiilor criptografice. Aceasta pentru ca în anii anteriori, experiența a dovedit nesiguranța metodelor de criptare, criptanaliza lor fiind uneori trivială (cifrul lui Cezar, Vigenere) sau uneori atinsă cu ceva mai mult efort precum Enigma și alte metode din cel de-al doilea război mondial.

Schemele de partajare sunt utilizate în aplicații precum: protejarea și recuperarea cheilor criptografice, vot electronic, certificate distribuite unor autorități, licitații on-line sau sisteme de stocare de lungă durată. [5]

O schemă de partajare reprezintă o metodă de a distribui un secret unor participanți, oferind fiecărui participant o componentă (share) astfel încât doar o submulțime de participanți pot recupera secretul inițial.

Cum volumul de datelor electronice este în continuă creștere, necesitatea stocării sigure a devenit extrem de necesară.

Date sensibile trebuie menținute secrete, câteodată pentru lungi perioade de timp (zeci de ani). Exemple includ informații militare, documete legale sau dosare medicale: informațiile militare pot fi clasificate zeci de ani, un testament trebuie menținut secret până la deschiderea sa oficială, un dosar medical necesită păstrare sigură (cel puțin) pe toată durata vieții pacientului.

TODO Notatii, PPT, uniform aleator, functie neglijabila

Notăm cu $s \leftarrow^R S$ faptul că s este ales în mod aleator uniform din mulţimea S.

Capitolul 1

Funcții Pseudoaleatoare

1.1 Introducere

O funcţie pseudoaleatoare (PRF) $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ este un algoritm polinomial determinist definit pe mulţimea cheilor $\mathcal{K}$ unde $\mathcal{X}$, $\mathcal{Y}$ reprezintă domeniul, respectiv codomeniul lui F. Primele construcţii au fost definite în 1986 pornind de la un PRG (generator pseudoaleator) sigur [6]. Această primitivă este folosită în special pentru generarea de chei în protocoale actuale precum TLS (fostul SSL) [7].

Un caz particular de PRF în care $\mathcal{Y} = \mathcal{X}$ şi F este bijectivă, sunt permutările pseudoaleatoare (PRP). Acestea au poprietatea utilă de a admite inversă, fiind utilizate direct în sisteme de criptare simetrice (ex. AES [8]).

Intuitiv, funcţia F poate fi evaluată eficient (în timp polinomial) în toate punctele $x \in \mathcal{X}$ din momentul în care o cheie $k \in \mathcal{K}$ este fixată. Din punct de vedere al securității, un adversar PPT (timp probabilistic polinomial) $\mathcal{A}$ nu poate face diferența între o instanţă aleatoare $F(k, \cdot)$, $k \in \mathcal{K}$ şi o funcție aleasă aleator din toate funcţiile posibile având domeniul $\mathcal{X}$, codomeniul $\mathcal{Y}$.

1.2 Definiții de Securitate

În procesul formalizării noţiunii de securitate *adaptivă* a unei funcții PRF F, vom avea în vedere un adversar PPT $\mathcal{A}$. În definiție sunt prezentate 2 experimente cu adversarul $\mathcal{A}$: $EXP(b)$, $b \leftarrow^R \{0,1\}$, iar avantajul lui va consta în abilitatea de a le distinge.

La începutul experimentului este apelată funcția $Setup(1^\lambda)$ care primește la intrare un parametru de securitate λ și întoarce o cheie secretă $k \leftarrow^R \mathcal{K}$. Fie $V, C \subseteq X$, 2 mulțimi de puncte (inițial vide) pentru care $\mathcal{A}$ cunoaște valorile $F(k,\cdot)$ trimise la $\mathsf{F.eval}$, respectiv la $\mathsf{F.challenge}$. De asemenea, $\mathcal{A}$ are acces la 2 tipuri de oracole:

- $\mathsf{F.eval}$: pentru un $x \in \mathcal{X}$ întoarce $F(k,x)$ dacă $x \notin C$ și actualizează mulțimea $V \leftarrow V \cup x$, altfel $\perp$ ($\notin \mathcal{Y}$).
- $\mathsf{F.challenge}$: pentru un $x \notin V$ întoarce $F(k,x)$ dacă $b = 0$ și $y \leftarrow^R \mathcal{Y}$ dacă $b = 1$ iar $C \leftarrow C \cup x$.

La finalul experimentului, $\mathcal{A}$ întoarce o valoare $b' \in \{0,1\}$.

Fie W_b evenimentul ca în urma eperimentului $EXP(b)$, b' să fie 1 la sfârșitul jocului. Notăm cu $AdvPRF_{\mathcal{A},F}(\lambda) = |Pr[W_0] - Pr[W_1]|$ avantajul lui A.

Definiția 1.2.1. *O funcție PRF F este sigură dacă pentru orice adversar PPT A, $AdvPRF_{\mathcal{A},F}(\lambda)$ este o funcție neglijabilă.*

1.3 Funcții Pseudoaleatoare Constrânse

O funcție cPRF (pseudoaleatoare constrânsă) $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ este o funcție PRF care deține *chei de constrângere* ce permit evaluarea funcției F în subdomenii $S \subseteq 2^{\mathcal{X}}$ menținând proprietățile de securitate ale funcțiilor PRF.

Definiția 1.3.1. *O funcție PRF $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ este constrânsă la o mulțime $\mathcal{S} \subseteq 2^{\mathcal{X}}$ dacă există o mulțime adițională de chei $\mathcal{K}_c$ și 2 algoritmi:*

- $\mathsf{F.constrain(k, S)}$*: un algoritm nedeterminist care pentru o cheie $k \in \mathcal{K}_c$ și o mulțime $S \subseteq \mathcal{S}$ întoarce o cheie $k_s \in \mathcal{K}_c$ pentru care se poate face*

evaluarea lui $F(k, x)$ pentru orice $x \in S$.

– $\mathsf{F.eval}(\mathsf{k_s}, \mathsf{x})$*: un algoritm determinist care pentru o cheie $k_s \in \mathcal{K}_c$ și un punct $x \in \mathcal{X}$ întoarce $F(k, x)$ dacă $x \in S$, sau $\perp$ $(\notin \mathcal{Y})$ dacă $x \notin S$*

Alte construcții pe baza cPRF definite în [9] și care vor fi folosite ulterior sunt: *stânga/dreapta PRF* - funcții cPRF ce admit evaluarea lor având partea stângă a domeniului fixată; *bit-fixate PRF* - funcții cPRF care pot fi evaluate în domenii corespunzătoare unei expresii având unii biți fixați. Autorii [9] menționează că aceste construcții sunt posibile în contextul funcțiilor multiliniare iar problema devine mai interesantă (deocamdată nerezolvată) în contextul LWE [10].

Definiția 1.3.2. *Fie o funcție $F : \mathcal{K} \times \mathcal{X}^2 \to \mathcal{Y}$ PRF. $\forall v \in \mathcal{X}$ o funcție stânga/dreapta admite 2 chei de constrângere ${k_v}^L$ și ${k_v}^R$ ce permit evaluarea funcției F în toate punctele $(v, x) \in \mathcal{X}^2$, respectiv $(x, v) \in \mathcal{X}^2$. Notăm această funcție cPRF stânga/dreapta cu $PRF^{L/R}$.*

Definiția 1.3.3. *Fie o funcție $F : \mathcal{K} \times \{0,1\}^n \to \mathcal{Y}$ PRF. $\forall v \in \{0,1,?\}^n$ o funcție bit-fixată admite o cheie de constrângere k_v ce permite evaluarea funcției F în toate punctele $x \in \{0,1\}^n$ ce satisfac modelul lui v. Notăm această funcție cPRF bit-fixată cu PRF^{bf}.*

Noțiunea de securitate pentru cPRF este asemănătoare cu cea pentru PRF: 2 experimente $EXP(0), EXP(1)$, doar că în acest caz este necesară existența unui oracol *F.constrain* pentru a permite adversarului să interogheze F conform definiției 1.3.1 [9].

La începutul fiecărui experiment este apelată funcția $Setup(\lambda)$ care întoarce o cheie secretă $k \leftarrow^R \mathcal{K}$. Inițial mulțimile $V, C \subseteq \mathcal{X}$ sunt vide iar adversarul $\mathcal{A}$ are acces la 3 oracole:

– $\mathsf{F.eval}$: pentru un $x \in \mathcal{X}$ întoarce $F(k, x)$ dacă $x \notin C$ și actualizează mulțimea $V \leftarrow V \cup x$, altfel $\perp$;

– $\mathsf{F.constrain}$: pentru o mulțime $S \in \mathcal{S}$ întoarce $F.constrain(k, S)$ dacă $S \cap C = \emptyset$ și actualizează mulțimea $V = V \cup x$, altfel $\perp$;

– $\mathsf{F.challenge}$: pentru un $x \notin V$ întoarce $F(k, x)$ dacă $b = 0$ și $y \leftarrow^R \mathcal{Y}$

dacă $b = 1$ iar $C \leftarrow C \cup x$;

La finalul experimentului, $\mathcal{A}$ întoarce o valoare $b' \in \{0, 1\}$.

Fie W_b evenimentul ca în urma eperimentului $EXP(b)$, b' să fie 1 la sfârșitul jocului. Notăm cu $AdvPRF_{\mathcal{A},F}(\lambda) = |Pr[W_0] - Pr[W_1]|$ avantajul lui A.

Definiția 1.3.4. *O funcție cPRF F este sigură dacă pentru orice adversar PPT $\mathcal{A}$, $AdvPRF_{\mathcal{A},F}(\lambda)$ este o funcție neglijabilă.*

Capitolul 2

NIKE

NIKE (Non Interactive Key Exchange - schimb de chei non-interactiv) reprezintă o primitivă criptografică importantă în care o mulțime de participanți pot stabili de comun acord o cheie comună fără ca aceștia să fie on-line în același timp. Un exemplu des întâlnit în zilele noastre este schimbul de chei Diffie-Hellman [11]. Construcția a fost apoi extinsă la 3 persoane de Joux folosind funcții biliniare [12] iar Boneh și Silverberg au generalizat la n persoane (multi-party) bazându-se pe funcții multi-liniare [13].

2.1 ID-NIKE

Criptografia bazată pe identitatea participanților a fost introdusă de Shamir în anul 1985 [14]. Aceasta presupune folosirea identității (email, nume, stradă), informație care îl identifică pe un utilizator într-un mod unic, fără să genereze o pereche de cheie publică, secretă. Principalul dezavantaj este necesitatea unei terțe autorizată care să administreze generarea de chei secrete, însă avantajul este utilizarea de mai puține resurse din partea participanților.

Așadar criptografia bazată pe ID este ideală într-un mediu în care utilizatorii dețin resurse computaționale limitate (rețea de routere).

ID-NIKE combină principiile oferite de NIKE cu ID-urile participanților.
Definiția 2.1.1. *Formal, ID-NIKE constă în* 3 *algoritmi:*

- *Setup*(λ)*: primește ca date de intrare parametrul de securitate* λ *și întoarce o cheie secretă* msk *împreună cu parametrii publici* $params$*;*
- *Extract*(msk, id_i)*: primește ca date de intrare cheia master secretă* msk*, identitatea* $id_i \in \mathcal{ID}$ *și întoarce cheia privată* sk_{id_i}*;*
- *KeyGen*($params, sk_{id_i}, id_j$)*: primește ca date de intrare parametrii publici* $params$*, o cheie secretă* sk_{id_i}*, o identitate* $id_j \neq id_i$ *și întoarce cheia comună* k_{id_i,id_j}*.*

Corectitudine

Este garantată dacă pentru orice pereche de participanți, $id_i, id_j \in \mathcal{ID}$, $id_i \neq id_j$ $KeyGen(params, sk_{id_i}, id_j) = KeyGen(params, sk_{id_j}, id_i)$.

Securitate

Amintim definiția de securitate adaptivă pentru ID-NIKE din [15].

Fie $EXP(b)$, $b \leftarrow^R \{0,1\}$, două experimente cu un adeversar PPT $\mathcal{A}$. La începutul fiecărui experiment $EXP(b)$ se execută $Setup(\lambda)$ oferind adversarului $\mathcal{A}$ $params$ și menținând cheia master msk privată. $\mathcal{A}$ are acces la 3 oracole:

- $Ext(id_i)$: primește ca date de intrare o identitate id_i, apelează $Extract(msk, id_i)$ și întoarce sk_{id_i};
- $Rev(id_i, id_j)$: primește ca date de intrare 2 identități id_i, id_j, $id_i \neq id_j$, apelează funcția $Extract(msk, id_i)$ și întoarce cheia comună k_{id_i,id_j} din $KeyGen(params, sk_{id_i}, id_j)$;
- $Test(id_i, id_j)$: primește ca date de intrare 2 identități id_i, id_j, $id_i \neq id_j$. Dacă $b = 0$ atunci întoarce $Rev(id_i, id_j)$, altfel întoarce $sk_{id_i,id_j} \leftarrow^R \mathcal{SKS}$ unde cu $\mathcal{SKS}$ am notat spațiul de chei comune posibile.

Pentru a elimina situația în care adersarul $\mathcal{A}$ câștigă trivial, acestuia nu îi este permis să interogheze Ext pentru o identitate în $Test$, precum și faptul că toate interogările Rev și $Test$ să fie distincte.

La finalul experimentului, $\mathcal{A}$ întoarce o valoare $b' \in \{0,1\}$.

Fie W_b evenimentul ca în urma eperimentului $EXP(b)$, b' să fie 1 la sfârșitul jocului. Notăm cu $AdvKE_{\mathcal{A}}(\lambda) = |Pr[W_0] - Pr[W_1]|$ avantajul lui A.
Definiția 2.1.2. *ID-NIKE este adaptiv sigură dacă pentru orice adersar PPT, $\mathcal{A}$ funcția $AdvKE_{\mathcal{A}}(\lambda)$ este neglijabilă.*

2.1.1 Construcția Boneh-Waters (2013)

În figura 2.1 prezentăm construcția Boneh-Waters bazată pe funcții constrânse stânga/dreapta [9].

$\mathsf{Setup}(\lambda)$: Fie $F : \mathcal{K} \times \mathcal{X}^2 \to \mathcal{Y}$ o funcție cPRF stânga/dreapta sigură, alege
în mod aleator $msk \leftarrow^R \mathcal{K}$ și întoarce parametrii publici $params$ corespunzători funcției F;

$\mathsf{Extract}(msk, id_i)$: calculează $\mathsf{F.constrain}(msk, \{(id_i, \cdot)\})$ pentru a obține $k^L_{id_i}$
și $\mathsf{F.constrain}(msk, \{(\cdot, id_i)\})$ pentru a obține $k^R_{id_i}$, apoi
întoarce $sk_{id_i} = (k^L_{id_i}, k^R_{id_i})$;

$\mathsf{KeyGen}(params, sk_{id_i}, id_j)$: întoarce $F(msk, (id_i, id_j))$ dacă $id_i < id_j$ și
$F(msk, (id_j, id_i))$ dacă $id_i > id_j$ (în ordine lexicografică).

Figura 2.1: Construcția Boneh-Waters [9]

Corectitudine

Reiese din funcția cPRF stânga/dreapta. Fie 2 participanți având identitățile id_i, id_j cu $id_i < id_j$. Participantul cu id_i folosește cheia $k^L_{id_i}$ iar cel cu id_j cheia $k^R_{id_i}$. În acest moment id_i poate calcula $F(msk, (id_i, \cdot))$ iar id_j, $F(msk, (\cdot, id_j))$ în acest mod stabilind cheia comună $F(msk, (id_i, id_j))$.

Securitate

Reiese din securitatea funcției cPRF stânga/dreapta. Dacă un adversar $\mathcal{B}$ împotriva unei funcții cPRF stânga/dreapta simulează perfect adeversarul $\mathcal{A}$ pentru construcția Boneh-Waters atunci demonstrația este completă. Așadar, luând în considerare cele 3 oracole din 2.1, în cazul in care $\mathcal{A}$ interoghează:

- $Ext(id_i)$: $\mathcal{B}$ interoghează F.constrain pentru toate S având valorile $(id_i, \cdot)$ și $(\cdot, id_i)$; întoarce rezultatul lui $\mathcal{A}$;
- $Rev(id_i, id_j)$: $\mathcal{B}$ interoghează F.eval pentru (id_i, id_j) dacă $id_i < id_j$, altfel interoghează pentru (id_j, id_i); întoarce rezultatul lui $\mathcal{A}$;
- $Test(id_i, id_j)$: $\mathcal{B}$ interoghează F.challenge și întoarce răspunsul lui $\mathcal{A}$.

2.2 Multi-Party ID-NIKE

Multi-Party (mai mulți membri) ID-NIKE reprezintă generalizarea protocolului ID-NIKE pentru cazul cu n participanți ($n > 2$). Mai întâi definim modelul de securitate pentru acest protocol după care introducem construcția multi-party bazată pe funcții cPRF bit-fixate din secțiunea 2.3 [3].

Definiția 2.2.1. *O schemă multi-party ID-NIKE este formată din* 3 *algoritmi:*

- *$Setup(\lambda, n)$: primește ca date de intrare parametrul de securitate λ și n, un număr întreg reprezentând numarul de participanți; calculează cheia secretă master msk și intoarce parametrii publici $params$;*
- *$Extract(msk, id_i)$: primește cheia secretă master msk, o identitate $id_i \in \mathcal{ID}$ și întoarce o cheie secretă sk_{id_i};*
- *$KeyGen(params, sk_{id_i}, \{id_l\}_{l=1,\dots,n})$: primește parametrii publici $params$, n identități distincte $id_1, id_2, \dots, id_n$, cheia secretă sk_{id_i} a identității id_i și întoarce cheia comună $sk_{id_1,id_2,\dots,id_n}$.*

Corectitudine

Este necesar ca toți participanții să obțină aceeași cheie; Pentru orice pereche $i, j, i \neq j$ și n identități distincte $id_1, id_2, \ldots, id_n$: $KeyGen(params, sk_{id_i}, \{id_l\}_{l=1,\ldots,n}) = KeyGen(params, sk_{id_j}, \{id_l\}_{l=1,\ldots,n})$.

Securitate

Definiția de securitate este similară ca în cazul ID-NIKE. La începutul fiecărui experiment $EXP(b)$ se execută $Setup(\lambda, n)$ iar parmetrii $params$ se fac publici. Un adversar $\mathcal{A}$ are acces la 3 tipuri de oracole:

- $Ext(id_i)$: pentru o identitate id_i apelează $Extract(msk, id_i)$ și întoarce sk_{id_i};
- $Rev(id_1, id_2, \ldots, id_n)$: pentru n identități $id_1, id_2, \ldots, id_n$ distincte, apelează $Extract(msk, id_i)$ pentru a afla sk_{id_i}; Întoarce $KeyGen(params, sk_{id_i}, \{id_l\}_{l=1,\ldots,n})$;
- $Test(id_1, id_2, \ldots, id_n)$: dacă $b = 0$ întoarce $Rev(id_1, id_2, \ldots, id_n)$; dacă $b = 1$ întoarce $sk_{id_1,id_2,\ldots,id_n} \leftarrow^R \mathcal{SKS}$.

Pentru a elimina situația în care adersarul $\mathcal{A}$ câștigă trivial, acestuia nu îi este permis să interogheze Ext pentru o identitate în $Test$, precum și faptul că toate interogările Rev și $Test$ trebuie să fie distincte. La finalul experimentului, $\mathcal{A}$ întoarce o valoare $b' \in \{0, 1\}$.

Fie W_b evenimentul ca în urma eperimentului $EXP(b)$, b' să fie 1 la sfârșitul jocului. Notăm cu $AdvKE_{\mathcal{A},n}(\lambda) = |Pr[W_0] - Pr[W_1]|$ avantajul lui A.
Definiția 2.2.2. *Multi-party ID-NIKE este adaptiv sigură dacă pentru orice adersar PPT, $\mathcal{A}$ funcția $AdvKE_{\mathcal{A},n}(\lambda)$ este neglijabilă.*

2.3 Contribuții

Pornind de la construcția Boneh-Waters ID-NIKE extindem protocolul la o schemă având n participanți, înlocuind funcția cPRF stânga/dreapta cu o funcție cPRF bit-fixată [3].

$\mathsf{Setup}(\lambda, n)$: Fie $F : \mathcal{K} \times \mathcal{X}^n \to \mathcal{Y}$ o funcție cPRF bit-fixată sigură, alege în mod aleator $msk \leftarrow^R \mathcal{K}$ și întoarce parametrii publici $params$ corespunzători funcției F;

$\mathsf{Extract}(msk, id_i)$: calculează:
$\mathsf{F.constrain}(msk, \{(id_i, \cdot, \ldots, \cdot)\})$ pentru a obține $k^1_{id_i}$,
$\mathsf{F.constrain}(msk, \{(\cdot, id_i, \ldots, \cdot)\})$ pentru a obține $k^2_{id_i}$ ...
$\mathsf{F.constrain}(msk, \{(\cdot, \ldots, \cdot, id_i)\})$ pentru a obține $k^n_{id_i}$,
apoi întoarce $sk_{id_i} = (k^1_{id_i}, k^2_{id_i}, \ldots, k^n_{id_i})$;

$\mathsf{KeyGen}(params, sk_{id_i}, \{id_l\}_{l=1,\ldots,n})$:
întoarce $F(msk, (id_{\pi(1)}, id_{\pi(2)}, \ldots, id_{\pi(N)}))$,
unde $id_{\pi(1)} < id_{\pi(2)} < \cdots < id_{\pi(N)}$ (în ordine lexicografică).

Figura 2.2: Construcția multi-party ID-NIKE [3]

Corectitudine

Este garantată din construcția funcției cPRF bit-fixate. Pentru a obține cheia comună, participantul cu identitatea id_i folosește $k^{\pi(i)}_{id_i}$ pentru a evalua funcția F.

Securitate

Reiese din securitatea funcției cPRF bit-fixate. Dacă un adversar $\mathcal{B}$ împotriva unei funcții cPRF bit-fixate simulează perfect adeversarul $\mathcal{A}$ pentru multi-party ID-NIKE atunci demonstrația este completă. Așadar, luând în considerare cele 3 oracole din 2.1, în cazul in care $\mathcal{A}$ interoghează:

- $Ext(id_i)$: $\mathcal{B}$ interoghează $\mathsf{F.constrain}$ pentru toate punctele din domeniul $\mathcal{X}^n$ care conțin pe id_i și întoarce rezultatul lui $\mathcal{A}$;
- $Rev(id_1, \ldots, id_n)$: $\mathcal{B}$ interoghează $\mathsf{F.eval}$ pentru $id_{\pi(1)}, id_{\pi(2)}, \ldots, id_{\pi(N)}$ și întoarce rezultatul lui $\mathcal{A}$;
- $Test(id_1, \ldots, id_n)$: $\mathcal{B}$ interoghează $\mathsf{F.challenge}$ și întoarce răspunsul lui

$\mathcal{A}$.

Dacă $\mathcal{A}$ reușește să diferențieze $F(msk, (id_{\pi(1)}, id_{\pi(2)}, \ldots, id_{\pi(N)}))$ de obiecte aleatoare din $\mathcal{Y}$ atunci $\mathcal{B}$ poate să rezolve $\mathsf{F.challenge}$ răspunzând cu același bit ca $\mathcal{A}$.

Capitolul 3

Scheme de Partajare

3.1 Introducere

O schemă de partajare constă în distribuirea unei informații secrete $\mathcal{S}$ la mai mulți participanți $\mathcal{P} = \{P_1, \ldots, P_n\}$ astfel încât oricare mulțime de participanți, 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 primește la intrare un secret S și un număr întreg m și întoarce un set de componente $s_1, s_2, \ldots, s_m$.
- $Rec(s_{i1}, s_{i2}, \ldots, s_{iq})$ este un algoritm care primește ca parametri de intrare o mulțime de componente și întoarce S dacă mulțimea $\{P_{i1}, P_{i2} \ldots P_{iq}\} \in \mathcal{A}$.

Majoritatea schemelor constau în mai multe etape precum:

- *Inițializare*. Presupune inițializarea variabilelor de mediu necesare.
- *Generare*. O entitate autorizată (numită dealer) $\mathcal{D}$ folosește algoritmul Gen pentru a genera componentele.
- *Distribuție*. Componentele sunt trimise participanților cu ajutorul unui mijloc de comunicare sigur, fără ca acestea sa fie vizibile unui atacator.

– *Reconstrucție.* Dându-se o mulțime de componente, se folosește algoritmul *Rec* pentru a recupera secretul $\mathcal{S}$.

Schemele de partajare se clasifică în funcție de cantitatea de informație secretă pe care o pot obține persoanele care nu fac parte din $\mathcal{A}$:

– *Sisteme perfecte de partajare*: componentele nu oferă nici o informație teoretică despre $\mathcal{S}$ indiferent de resursele computaționale.

– *Sisteme statistic sigure*: o fracțiune de informație este dezvaluită despre $\mathcal{S}$, independent de puterea computațională a adversarului.

– *Sisteme computațional-sigure de partajare*: se bazează pe faptul că reconstituirea lui $\mathcal{S}$ se reduce la o problema *dificilă* (spre exemplu, problema Diffie-Hellman [16]) în lipsa unor informații oferite doar grupului de acces $\mathcal{A}$ [5].

Primele scheme de partajare au fost dezvoltate independent de Shamir și Blakley în 1979 [17, 18].

Denumite și scheme majoritare (k, n), acestea rezolvau cazul în care oricare grup de participanți cu un număr mai mare sau egal decât k (mărimea pragului) pot reconstitui secretul $\mathcal{S}$ din componentele primite de la dealerul $\mathcal{D}$. Dacă schema este perfect *sigură* atunci oricare grup cu un număr de participanți mai mic decât k nu obține nici o informație despre $\mathcal{S}$.

Schemele majoritare (spre exemplu schema Shamir) sunt insuficiente pentru a permite partajarea pentru anumite structuri de acces. Considerăm cazul în care dorim să partajăm un secret între 4 participanți: P_1, P_2, P_3, P_4 astfel încât $\{P_1, P_2\}$ și $\{P_3, P_4\}$ să fie singurele mulțimi autorizate pentru reconstrucția 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 mulțimi cu un număr de 2 participanți 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 și Nishizeki, realizând o generalizare a schemei Shamir [19]. Benaloh și Leichter au demonstrat că schemele de partajare de tip prag nu pot fi folosite pe structuri general monotone (familie de submulțimi ale lui $\mathcal{P}$ cu proprietatea că dacă $A \in \mathcal{A}$ și $A \subset A'$, atunci $A' \in \mathcal{A}$) obținând o construcție mai eficientă ca Ito et. al din punct de vedere al numărului de componente distribuite participanților [20].

3.2 Schema unanimă

Presupunând că împărțim un secret $\mathcal{S}$ la n participanți astfel încât $\mathcal{S}$ să poată fi recuperat doar dacă toți cei n participanți își combină componentele pe care le dețin. Metoda este echivalentă cu o schemă (n, n) majoritară. Un exemplu este schema introdusă de Karin, Greene și Hellman (Fig.3.1) [21].

Inițializare:
- Fie $S \in Z_q$ unde $q > 1$ și q prim;
- Fie n numărul de participanți;

Generare: Dealerul $\mathcal{D}$:
- Alege $n-1$ valori aleatoare $s_i \leftarrow^R Z_q$, $i \in \{1, 2, \ldots, n-1\}$;
- $s_n = S + \sum_{i=1}^{n-1} s_i \pmod q$;

Distribuție: Dealerul $\mathcal{D}$:
- transmite în mod sigur participantului P_i componenta s_i, $i \in \{1, 2, \ldots, n\}$;

Reconstrucție: Cei n participanți:
- Calculează $S = \sum_{i=1}^{n} s_i \pmod q$.

Figura 3.1: Schema unanimă [21]

3.3 Schema Shamir

Schema Shamir oferă mai multă flexibilitate decât schema unanima prin faptul ca oricare k (sau mai mulți) participanți din cei n pot recupera $\mathcal{S}$, însă mai puțin de k participanți nu obțin nicio informație despre $\mathcal{S}$. Schema Shamir este, astfel, 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 de grad $k-1$ ce 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 soluții. Evident, orice submulțime de valori s_i de

mărime egală cu k este suficientă și necesară pentru a reconstrui polinomul f. După interpolarea componentelor deținute de cel puțin k dintre participanți, secretul $\mathcal{S}$ se află în $f(0)$ (Fig. 3.2) [18].

Pentru un atacator care deține chiar și $k-1$ valori s_i, acesta nu determină nimic despre $\mathcal{S}$, spațiul de soluții posibile fiind identic cu situația în care deține 0 componente.

Inițializare:
- Fie $S \in Z_q$ unde $q > 1$ și q prim;
- Fie n numărul de participanți a.î. $q > n$;
- Fie k numărul minim de componente puse în comun pentru a determina $\mathcal{S}$;

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

Distribuție: Dealerul $\mathcal{D}$:
- Transmite participantului P_i componenta s_i, $i \in \{1, \ldots, n\}$;

Reconstrucție: Orice mulțime cu dimensiunea k (sau mai mare) de participanți distincți $P_1, P_2, \ldots, P_k$:
- Interpolează punctele s_i pentru a obține polinomul f:

$$f(x) = \sum_{i=1}^{k} s_i \prod_{1 \leq j \leq k, j \neq i} \frac{x - x_j}{x_i - x_j} \tag{3.1}$$

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

Figura 3.2: Schema Shamir [18]

3.4 Schema Ito, Saito și Nishizeki

În continuare vom descrie modalitatea de distribuire a componentelor de la care au pornit Ito, Saito și Nishizeki pentru ca schema sa aibă o structură de acces $\mathcal{A} \subseteq 2^P$ (submulțime a setului de participanți) monotonă (i.e. $\forall A \in \mathcal{A}, A \subseteq A' \Rightarrow A' \in \mathcal{A}$) [19]. Folosind construcția unei scheme majoritare (k, n) autorii au reușit să descrie elementele din $\mathcal{A}$ folosind rezultatul unei reuniuni de mulțimi de componente cu un număr de elemente mai mare sau egal decât k (Fig. 3.3) [19]. Notația $x : Pr$, 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 funcției *Assign* din figura Fig. 3.3 este următorul: pentru fiecare mulțime minimală $A \in \mathcal{A}$ ($\forall B \in \mathcal{A}$, $B \neq A, B \not\subset A$) se folosește o schemă de partajare unanimă $(|A|, |A|)$ a lui $\mathcal{S}$ pentru participanții din A.

Exemplu 3.4.1. *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* $(3, 3)$

Participanții primesc in felul următor componentele:

- $P_1 : \{s_1, s_3\}$;
- $P_2 : \{s_2\}$;
- $P_3 : \{s_4\}$;
- $P_4 : \{s_5\}$;

3.5 Schema Feldman

Uneori, canalul de comunicaţie prin intermediul căruia dealerul $\mathcal{D}$ transmite componentele nu este întotdeauna sigur. Din fericire, participanții pot constata cu ușurință integritatea componentelor primite folosind scheme de

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

Generare: Dealerul $\mathcal{D}$:
- Alege n valori distincte $x_i \leftarrow^R Z_q$, $i = 1, 2, \ldots, n$;
- Alege $a_i \leftarrow^R \mathcal{C} \setminus \{0\}$, $i \in \{1, 2, \ldots, k-1\}$, $a_{k-1} \neq 0$;
- Construiește 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 funcția $Assign : P \rightarrow 2^Q$:
  - $Assign(P_i) = D_i$ $1 \leq i \leq n$
  - $\mathcal{A} = \left\{ Q \subseteq Shares : \left| \bigcup_{P_i \in Q} Assign(P_i) \right| \geq k \right\}$;

Distribuție: Dealerul $\mathcal{D}$:
- Transmite participantului P_i componenta $Assign(P_i)$, $i \in 1, 2, \ldots, n$;

Reconstrucție: Participanții din structura de acces $\mathcal{A}$:
- Procedează identic ca în schema Shamir.

Figura 3.3: Schema Ito, Saito, si Nishizeki [19]

verificare. Primele direcții către astfel de construcții au fost oferite în anul 1985 de către Chor et al [22].

În continuare vom exemplifica o schemă de verificare non-interactivă precum cea a lui Feldman, construită pe baza schemei Shamir. (Fig. 3.4) [23]. Spre deosebire de schema Shamir, securitatea schemei Feldman este una computațională și deține un model de adversar pentru care problema logaritmului discret este dificilă.

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

Inițializare: Folosind notațiile din Fig.3.2, considerăm polinomul f generat în urma Schemei Shamir.

– Se alege un $g \in Z_q$ generator al lui Z_q a.î. este dificil de calculat $\log_g x$

Generare: Dealerul $\mathcal{D}$:

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

Distribuție: Dealerul $\mathcal{D}$:

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

Figura 3.4: Schema Feldman [23].

$$g^{s_i} = c_0 c_1^i c_2^{i^2} \ldots c_t^{i^k} = \prod_{j=0}^{k} c_j^{i^j} = \prod_{j=0}^{k} g^{a_j i^j} = g^{\sum_{j=0}^{k} a_j i^j} = g^{f(i)} \qquad (3.2)$$

Capitolul 4

Sisteme de stocare de lungă durată

4.1 Introducere

Un sistem de stocare, în general, trebuie să satisfacă cel puțin următoarele 3 condiții:

- Disponibilitatea: Informația trebuie să rămână accesibilă tot timpul, chiar și în prezența erorilor de tip hardware.
- Integritatea: Abilitatea sistemului de a răspunde cererilor într-un mod care garantează corectitudinea lor.
- Confidențialitatea: O persoană care nu face parte din grupul de acces să nu obțină permisiunea de a afla informații de orice fel despre datele existente in sistem.

4.2 Criptare vs. scheme de partajare

Una dintre soluțiile existente pentru a construi acest sistem este criptarea datelor folosind o cheie înainte de inserarea lor in spațiul de stocare. în

momentul în care un utilizator autorizat dorește să efectueze o citire a unor date, întrebuințează cheia potrivită pentru a le decripta.

În practică, există algoritmi de criptare eficienți precum AES însă aceștia nu garantează confidențialitatea datelor în cazul în care apare un adversar fără o limită computațională.

Un dezavantaj al criptării este adminstrarea cheilor, standardele de securitate schimbându-se în fiecare an. De fiecare dată când cheile sunt înnoite atunci este necesară recriptarea datelor de pe fiecare bază de date. Cu cât disponibilitatea este mai mare - volumul de date crește - recriptarea informației devine o operație foarte costisitoare.

Majoritatea tehnicilor de criptarea se bazează pe dificultatea factorizării unui număr sau cea a calculării logaritmului discret însă o dată cu posibila dezvoltare a calculatoarelor cuantice aceste probleme nu vor mai fi atât de dificile [24]. Pentru schemele de partajare, spre desoebire de criptare, avantajul unui adversar nu depinde de puterea sa computațională, acestea garantând securitatea teoretică, precum schema Shamir [18].

Dezavantajul schemelor generale de partajare este dimensiunea componentelor, exponențială în funcție de numărul de participanți [25].

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

O alternativă la soluția cu criptare care asigură atât confidențialitate cât și redundanța necesară este intrebuințarea sistemelor de stocare de lungă durată bazate pe scheme de partajare [2, 26, 27].

4.3.1 PASIS

PASIS este o soluție pentru un sistem descentralizat care oferă beneficii precum securitate, redundanță a datelor si auto-întreținere [26].

Structurile descentralizate împart informația la mai multe noduri folosind scheme de redundanța precum "Redundant Array of Independent Disks"

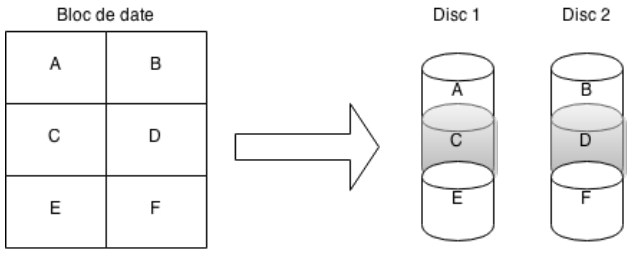

Figura 4.1: Procesul de striping aplicat unui bloc de date

(RAID) pentru a asigura performanța, scalabilitatea sistemului dar și integritatea datelor [28]. RAID reprezintă o tehnologie ce combină 2 concepte ortogonale precum data striping (aranjarea datelor pe discuri multiple într-o manieră secvențială - Fig 4.1) și redundanță pentru o disponibilitate ridicată [29].

PASIS folosește schemele de partajare pentru a distribui informația nodurilor de stocare dintr-o rețea. Aceasta introduce *agenți* pe partea clientului pentru a scrie sau șterge date din noduri, dar și agenți pentru mentenanță. Componentele obținute în urma partajării unui fișier sunt stocate in rețea cu ajutorul agenților (Fig. 4.2). Pe lângă conținutul brut al componentelor se adaugă metadate pentru a reține adresa nodului din rețea la care au fost stocate dar și noua denumire cu care este salvată in rețea.

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

- Caută numele celor n componente care alcătuiesc fișierul într-un serviciu care listează toate datele.
- Inițiază cereri de citire la cel puțin k din cele n noduri.
- în caz ca acesta nu primește cel puțin k răspunsuri se întoarce la pasul anterior încercând interogări la noduri diferite.
- Reconstituie fișierul obținut din cele k componente.

Operația de scriere este similară cu cea de citire, aceasta oprindu-se atunci când în cel puțin $n - k + 1$ noduri s-au stocat cu succes componente.

Autorii specifică soluții pentru auto mentenanța sistemului cu ajutorul resurselor

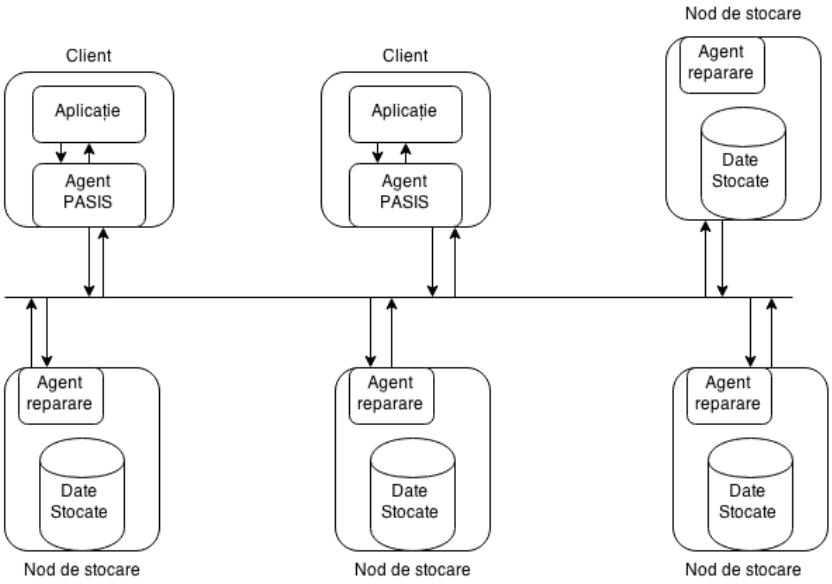

Figura 4.2: Arhitectura PASIS cu 4 noduri și 2 clienți [26].

umane prin monitorizarea periodică stării sale, folosind log-uri sau ajustarea parametrilor din cadrul schemei de partajare.

4.3.2 GridSharing

În 2005, Subbiah și Blough propun o nouă abordare pentru a construi un sistem de stocare securizat și tolerant la erori numit GridSharing [27].

Subbiah și Blough folosesc un sistem care înlocuiește schemele de verificare cu o schemă de partajare unanimă XOR (considerăm cazul $q = 2$ în Fig. 3.1) pentru a păstra securitatea construcției. 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 validității va fi stabilită în funcție de numărul de servere pe care le conțin.

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

– Abandonări: un server este *abandonat* dacă nu mai raspunde vreunui mesaj din rețea și s-a oprit din a mai efectua vreo operație.

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

– Scurgeri de informații: serverul execută protocoalele corect dar e posibil ca un adversar să fi obținut 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 folosește vulnerabilitățile cu intenția de a *învăța* din informații.

Arhitectura GridSharing constă în N servere, unde cel mult c servere pot fi abandonate, b servere bizantine și l cu scurgeri de informații. Cele N pot fi aranjate într-un grid cu r linii și N/r coloane (considerăm pentru simplitate că $N \pmod r = 0$).

Caracteristicile modelului bizantin și cel specific scurgerilor de informații permit dezvăluirea componentelor unui adversar de pe cel mult $l + b$ servere.

Exemplu 4.3.1. *Notăm* $\binom{x}{y}$ *fiind combinări de* x *elemente grupate câte* y*. Considerăm că împărțim un secret* $\mathcal{S}$ *la* 4 *linii (participanți) astfel încât sistemul să permită* 2 *componente de tip* b*,* 1 *componentă de tip* l *și* 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

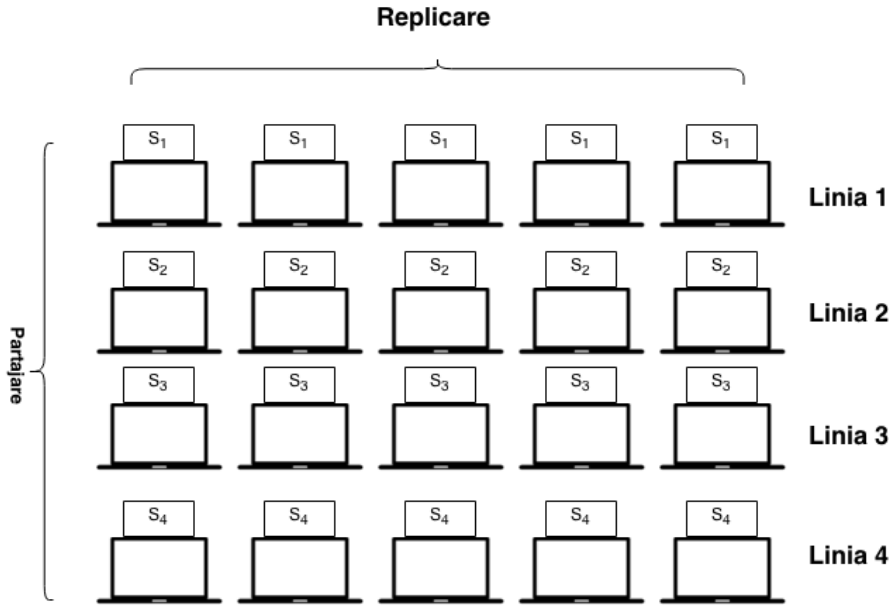

Figura 4.3: GridSharing cu 4 linii, 20 servere dintre care 2 bizantine, 1 cu scurgeri de informații [27].

4.3.3 POTSHARDS

în 2007 este propus un nou sistem care combină caracteristicile PASIS și GridSharing adăugând posiblitatea de migrare a datelor la noduri noi: POTSHARDS (Protection Over Time, Securely Harboring And Reliably Distributing Stuff) [2]. Este de menționat faptul că autorii acestui sistem au publicat un articol în care au evidențiat vulnerabilități la numeroase sisteme de stocare (printre care și cele descrise în Secțiunile 4.3.1, 4.3.2) [30].

POTSHARDS poate fi gândit ca o aplicație pe partea clientului care comunică cu o mulțime de noduri (arhive) independente, folosind reconstrucția componentelor într-un mod securizat și semnături algebrice pentru a asigura un grad ridicat de păstrare a integrității fișierelor [31].

Confidențialitatea este asigurată prin adoptarea unei scheme de partajare

XOR unanimă, la fel ca în GridSharing.

Ca prim pas, POTSHARDS preprocesează fișierul într-un obiect, partajează obiectul în fragmente la care adaugă meta-date, numite *shards* (Fig. 4.4) [2]. Acestea sunt trimise apoi arhivelor independente, fiecare având propriul domeniu de securitate, localizate in *regiuni*. Pentru a reconstitui cu succes informația inițială, meta-datele shard-urilor conțin detalii despre structura pointerilor aproximativi, indicând regiunea în care se află următorul shard. Pointerii aproximativi sunt folositi pentru a reconstitui întreaga arhivă doar din shard-uri.

Procesul de fragmentare a datelor este prezentat in Fig. 4.5.

Pentru ca reconstituirea unui fișier sa fie fezabilă unui utilizator, acestuia îi este întoarsă o listă cu locațiile exacte shard-urilor corespunzătoare. Obținerea unui shard de către un atacator nu este fezabilă, 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 distribuite într-un mod cât mai uniform [2].

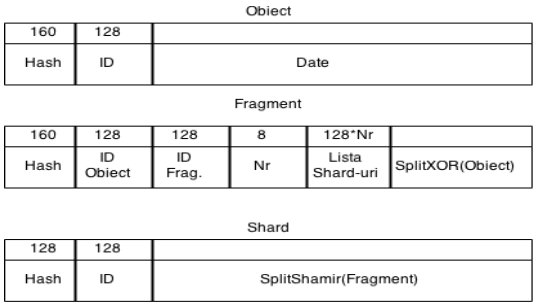

Figura 4.4: Entități de date in POTSHARDS. Nr e numărul de shard-uri produse de un fragment. $SplitXOR$ reprezintă o componentă rezultată în urma partajarii unanime XOR. Analog $SplitShamir$ reprezintă o componentă rezultată în urma partăjarii folosind schema Shamir [2]

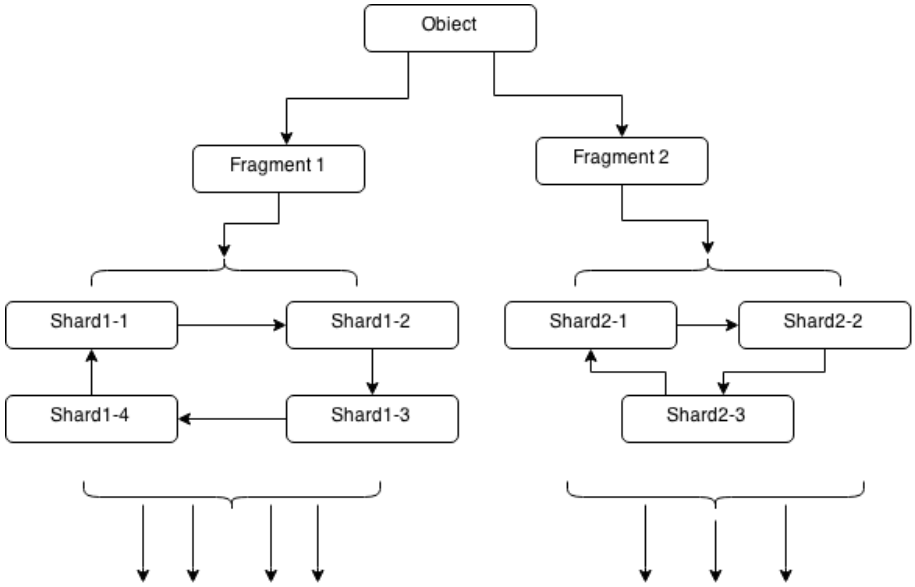

Figura 4.5: Distribuirea unui obiect in POTSHARDS

4.4 Sistemul de stocare Alouneh et al. (2013)

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 esențiale mai târziu în menținerea securității [32].

Pentru stocarea unui fișier în sistem (abordând filozofia majorității sistemelor de operare - orice este un fișier), o aplicație client îl împarte în blocuri de octeți de lungime k. Pentru fiecare bloc, octeții devin coeficienții unui polinom f de grad k, componenta corespunzătoare participantului i va fi reprezentată de valoarea lui $f(i)$, $i = \{1, 2, \ldots, n\}$. Menționăm că toate operațiile 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 4.6.

Exemplu 4.4.1. *Vom exemplifica modul de calcul în* $GF(256)$ $(mod\ g(x))$ *unde* $g(x) = x^8 + x^4 + x^3 + 1$. *Luăm polinomul* $f(x) = 10 + 15x$ *corespunzător*

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

Date de ieșire: n fișiere binare distribuite la noduri din rețea;

Procesarea componentelor: Aplicația existentă pe partea clientului:
- Dacă $\mathcal{S}$ nu are o lungime divizibilă cu k:
  - Concatenează la sfârsitul lui $\mathcal{S}$ octeți 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:
  - Construiește 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 $1 \leq i \leq n$;

Distribuție: Aplicația la nivelul clientului:
- Distribuie componenta $f(i)$ nodului din rețea cu indicele i:

Figura 4.6: Schema Alouneh et al. - Generare [32]

unui fișier format din octeții (în această ordine) 10 15.

$$\begin{aligned} f(01)\ (mod\ g(x)) &= 10 + 15\ (mod\ g(x)) \\ &= (x^4) + (x^4 + x^2 + 1)\ (mod\ g(x)) \\ &= x^2 + 1 = 000000101_{(2)} \\ &= 05_{(16)} \end{aligned} \tag{4.1}$$

$$\begin{aligned} f(02)\ (mod\ g(x)) &= 10 + 15 \cdot 02\ (mod\ g(x)) \\ &= (x^4) + (x^5 + x^3 + x)\ (mod\ g(x)) \\ &= 00111010_{(2)} \\ &= 3A_{(16)} \end{aligned} \tag{4.2}$$

Reconstituirea unui fișier (Fig. 4.7) se realizează din orice mulțime de componente A de dimensiune minim k folosind interpolarea Lagrange (utilizată și în cadrul schemei Shamir):

$$f(x) = \sum_{i \in A} f(i) \prod_{j \in A, j \neq i} \frac{x - j}{i - j} \tag{4.3}$$

Exemplu 4.4.2. *Vom exemplifica interpolarea conform ecuației 4.3 folosind componentele (4.1) și (4.2) calculate în exemplul anterior:*

$$\begin{aligned} f(x) &= 05(x-02)(01-02)^{-1} + 3A(x-01)(02-01)^{-1} \\ &= 05(x-02)03^{-1} + 3A(x-01)03^{-1} \\ &= F6(05+3A)x + F6(05\cdot 02 + 3A\cdot 01) \\ &= F6\cdot 3F\cdot x + F6\cdot 30 = 15x + 10 \end{aligned} \tag{4.4}$$

Noutatea arhitecturii constă in diminuarea redundanței componentelor la un factor de k, spre deosebire de sistemele descrise in Secțiunile 4.3.1, 4.3.2 și 4.3.3. Reducerea spațiului ocupat de componente este datorat înlocuirii coeficienților generați aleator din schema Shamir cu octeții din fișierul ce va fi partajat, fiecare componentă având nevoie de $1/k$ din dimensiunea fișierului $\mathcal{S}$.

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.

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

Date de ieșire: Fișierul binar original $\mathcal{S}$;

Reconstrucție: Aplicația existenta pe partea clientului:
- Repetă pentru fiecare bloc al lui $\mathcal{S}$:
  - Calculează prin interpolare coeficienții 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
- șterge octeții de la sfârșitul fișierului adăugați la generare.

Figura 4.7: Schema Alouneh et al. - Reconstrucție [32]

4.5 Contribuții

În [4] am analizat sistemul Alouneh et al. prezentat în Secțiunea 4.4. Am indentificat erori majore ale acestuia și am implementat sistemul descris de autori pentru a demonstra practic, nu doar teoretic, anumite greșeli pe care le vom evidenția în următoarele secțiuni.

4.5.1 Vulnerabilități evidențiate

Spre deosebire de schema Shamir, unde coeficienții (cu excepția termenului liber, care este egal cu secretul) sunt aleși într-un mod aleator uniform, sistemul propus de Alouneh et al. folosește o schemă modificată Shamir în care coeficienții sunt extrași din conținutul fișierelor originale. Alegerea este motivată de faptul ca mulțimea componentelor și efortul computațional depus pentru generarea coeficienților se reduce la un factor de k ori, spre deosebire de schema Shamir.

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

- Detectarea tipului unui fișier
- Detectarea conținutului unui fișier

Deoarece Alouneh et al. nu menționeaă o metodă de padding, am indicat că un atac bazat pe felul în care se realizează completarea fișierului (padding) $\mathcal{S}$, înainte de partajarea sa poate fi fezabil, în condițiile în care s-a demonstrat că această alegere este esențială în păstrarea securității [33].

Detectarea tipului de fișier

În tehnologia informației, la inceputul fiecărui fișier se află o secvență de octeți (denumită semnătură sau antet) cu rolul de a identifica tipul acestuia. Tabelul 4.1 indică 7 din cele mai uzuale antete.

Se consideră cazul în care partajarea unui fișier *pdf* se face cu ajutorul sistemului descris in Secțiunea 4.4, folosind $k \leq 4$. Polinomul corespunzător $f(x)$ va fi intotdeauna același. Presupunând ca numerotarea nodului i este aceeași, putem determina cu ușurință dacă este stocat un fișier *pdf* fără a lua în calcul conținutul fișierului. Acest atac este fezabil deorece valoarea lui k este publică iar i poate să 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 fișier.

Pentru a exemplifica, un adversar poate distinge cu probabilitate ridicată între fișierele *doc, gif, pdf, png, rar, wav* și *zip*. În Tabelul 4.2 avem generate componentele pentru $k = 2$ și $n = 5$. Dacă un adversar descoperă valoarea primului nod iar prima componentă este 14 atunci acesta știe cu certitudine că aceasta corespunde un fișier *gif*. Dacă obține accesul nodului 4 și citește valoarea 205 atunci știe că fișierul este de tipul *rar*. Daca citește valoarea 27 de pe primul nod atunci știe că poate fi un *wav* sau *zip*; apoi poate să distingă cele 2 fișiere cu probabilitate 1 dacă dezvăluie o singură valoare de pe celelalte noduri (2, 3, 4 sau 5) pentru că valorile sunt distincte.

Detectarea de conținut

Multe documente urmează un anumit tipar precum contracte, chitanțe, bonuri fiscale sau curriculum vitae. Deoarece majoritatea conținutului rămâne neschimbat, există o probabilitate destul de mare ca multe componente sa aibă aceeași valoare. O dată ce un adversar reușește să determine componentele unui nod, poate determina prin analogie tipul de conținut al fișierului original.

Fișierele vulnerabile sunt cele care conțin o secvență de octeți periodică (imagini cu un pattern repetitiv) sau cele care au preponderent octeți nuli (valoarea componentelor asociată majorității blocurilor fiind 0). Spre deose-

bire de prima metodă, aceasta nu necesită compararea cu un al 2-lea fișier, tratând componentele de sine stătător. Multiple componente identice indică existența unor secvențe repetitive în conținutul fișierului.

4.5.2 Implementare și rezultate practice

Pentru a arăta aplicabilitatea rezultatelor în practică, am implementat propunerea descrisa in Secțiunea 4.4 și 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, permițând programatorilor să exprime concepte în mai puține linii de cod față de C++ sau Java și este disponibil sub licență open-source [34]. Pentru a realiza comunicarea între procese am folosit Cerealizer iar distribuția componentelor a fost generată grafic cu ajutorul pachetului Matplotlib [35, 36]. De asemenea am folosit sistemul de versionare Git iar în prezent codul folosit este găzduit de GitHub [37, 38].

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

Menționăm această metodă doar pentru completitudine, aceasta neafectând rezultatele, care consideră doar secvențele de octeți din antet sau de la începutul fișierului.

Detectarea tipului de fișier

Extindem analiza făcută în Secțiunea 4.5.1 asupra tipurilor de fișiere din Tabelul 4.1 pentru $k = 2$ și creștem valoarea indicelui i până când 2 componente devin egale. Fie f_l polinomul de gradul 1 asociat primului bloc al fișierului aflat pe linia l. Analog f_c polinomul de gradul 1 asociat primului bloc al fișierului aflat pe coloana c.

Tabelul 4.3 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 4.3 observăm valoarea 0 pentru perechea (wav, zip) pentru că $f_{wav}(1) = f_{zip}(1) = 27$ (Tabelul 4.1).

Tabelele 4.4 și 4.5 prezintă rezultatele pentru $k = 3$ și $k = 4$. Pentru $k \geq 5$ este nevoie de un antet cu mai mult de 4 octeți.

Considerăm metoda de distribuire descrisă exact ca în Tabelul 4.6, și anume nodul de stocare i primește componenta $f(i)$. Dacă un atacator preia controlul nodului cu valoarea i, acesta poate face distincția între tipul a 2 fișiere partajate cu probabilitate 1 dacă i e mai mic decât valoarea afișată în tabel.

Spre exemplu, un adversar care obține accesul unui singur nod de stocare, iar partajarea a fost făcută pentru $k = 2$ poate distinge cu probabilitate 1 între un fișier *doc* sau *pdf* dacă $i \leq 194$. Deoarece $n > 194$ nu se întâmplă în practică, acest atac funcționează. Faptul că multe valori sunt ridicate și majoritatea sunt -1 (Tabelul 4.4), indică un nivel scăzut de securitate al schemei. În cazul celor cu -1 un adversar va câștiga întotdeauna cu probabilitatea 1, indiferent de indicele nodului de stocare pe care îl obține.

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

Detectarea de conținut

Considerăm scenariul pentru detectarea conținutului unui fișier, demonstrând practic cum un adversar poate să facă diferența între 2 componente stocate pe același nod aparțin unor documente similare. Atacul presupune accesul la un singur nod, indiferent de mărimea pragului k.

Pentru experiment am ales 3 fișiere *pdf* disponibile online la [39]:

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

Primele două fișiere reprezintă șablonul pentru un CV european în limba bulgară, respectiv daneză. Observăm că nu doar conținutul șabloanelor diferă, dar și limba în care au fost traduse. Cel de-al treilea fișier este complet diferit de primele două, fiind un document personal în limba română, folosit pentru înregistrarea cunoștiintelor dobândite într-o țară europeană.

Pentru a arăta vulnerabilitatea sistemului Alouneh et al. [32], partajăm cele 3 fișiere folosind schema $(2, 4)$. Experimentul reprezintă o implementare practică a metodei prezentate in Fig. 4.6.

Fig. 4.8, 4.9, 4.10, 4.11 conțin 2000 de componente generate pentru cele 3 fișiere, stocate în baza de date corespunzătoare nodurilor $1, 2, 3$ și 4. Pre-

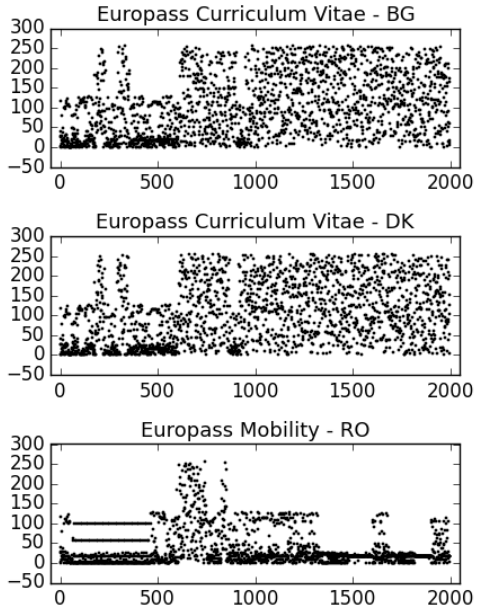

Figura 4.8: Nod 1: Graficul componentelor pentru primele 2000 de blocuri

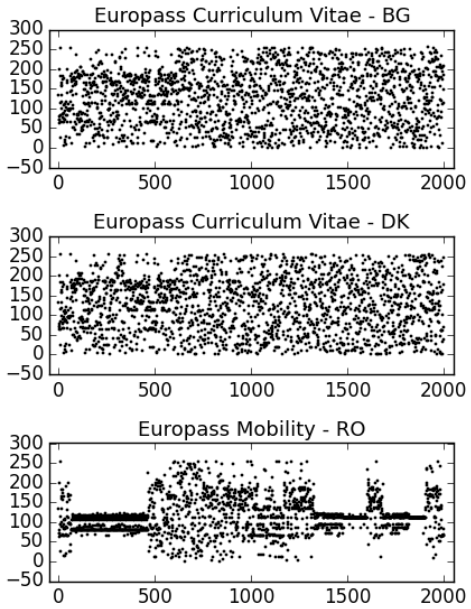

Figura 4.9: Nod 2: Graficul componentelor pentru primele 2000 de blocuri

supunem că indicele i este fixat, polinoamele fiind evaluate în același nod i. Un adversar care obține componentele unui nod poate deduce cu ușurință graficul lor.

Considerăm că un adversar reușește să obțină componentele din Fig. 4.8. Poate deduce ușor că primele două fișiere au un număr mare de componente identice iar cel de-al 3-lea este diferit de primele.

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

Considerăm în continuare un alt scenariu, în care partajăm o imagine pentru care informația se repetă. Fie imaginea din Fig. 4.12 partajată folosind schema din Secțiunea 4.4, având 4 noduri iar informația stocată pe cel puțin 2 din ele pot reconstitui imaginea partajată inițial. 4.6. La fel ca în exemplul

precedent, Fig. 4.13, 4.14, 4.15, 4.16 reprezintă grafic componentele imaginii din Fig. 4.12. Un adversar constată cu ușurință prin obținerea informațiilor stocate aflate pe un singur nod (indiferent de mărimea pragului k) că fișierul original conține un pattern care se repetă.

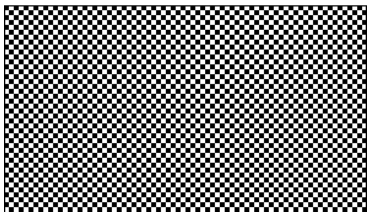

Figura 4.12: Imagine conținând pattern-uri

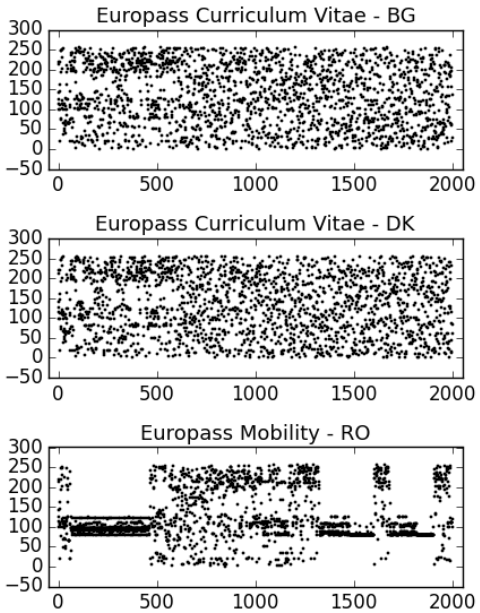

Figura 4.10: Nod 3: Graficul componentelor pentru primele 2000 de blocuri

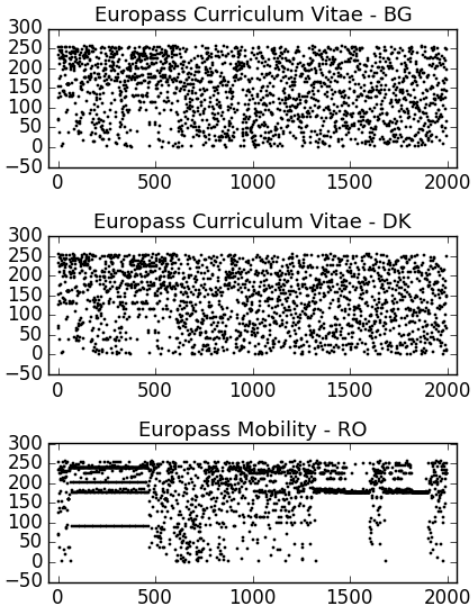

Figura 4.11: Nod 4: Graficul componentelor pentru primele 2000 de blocuri

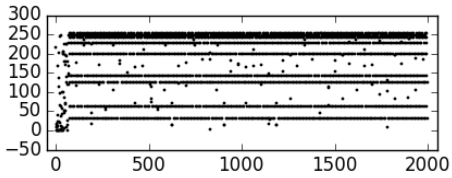

Figura 4.13: Nod 1: Graficul componentelor pentru primele 2000 de blocuri

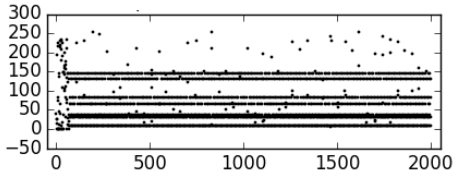

Figura 4.14: Nod 2: Graficul componentelor pentru primele 2000 de blocuri

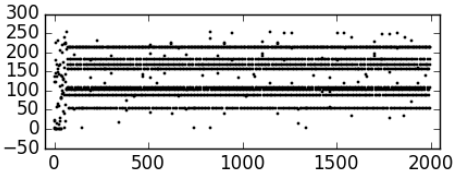

Figura 4.15: Nod 3: Graficul componentelor pentru primele 2000 de blocuri

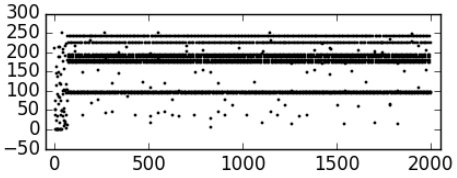

Figura 4.16: Nod 4: Graficul componentelor pentru primele 2000 de blocuri

Tabel 4.1: Semnături de fișiere

Tip de fișier	Primii 4 octeți			
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 4.2: Componentele primului bloc ($k = 2$)

Tip fișier	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

Tabel 4.3: Indicele maxim i a.î. componentele primului bloc sa fie distincte($k = 2$)

Tip Fișier	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 4.4: Indicele maxim i a.î. componentele primului bloc sa fie distincte($k = 3$)

Tip Fișier	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 4.5: Indicele maxim i a.î. componentele primului bloc sa fie distincte($k = 4$)

Tip Fișier	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	-

Capitolul 5

Implementare alternativă a sistemelor de stocare

Motivația principală vine din eliminarea unor vulnerabiltăți prezentate în Secțiunea 4.4 dar și o încercare de creștere a uzabilității schemelor de partajare, crearea unei interfețe intuitive, ușor de folosit.

Uzabilitatea a fost întotdeauna o barieră care rareori a fost depășită; dezvoltarea de unelte care pot fi ușor folosite de utilizatorii generali, păstrând ascuns stratul de criptografie din spatele lor dar menținând o securitate ridicată [40].

Conform [40], când construim arhitectura unei aplicații avem la îndemână următoarele alegeri când facem referire la uzabilitate și securitate:

1. Uzabilitatea și securitatea vor avea aceeași importanță;
2. Securitatea va fi tratată pe primul plan, lăsând un loc mai mic uzabilității;
3. Uzabilitatea va fi tratată pe primul plan, în dezavantajul securității;
4. Securitatea va fi un produs separat, incorporată într-un mod natural în aplicație fără a compromite design-ul ei.

Pentru a vedea de ce uzabilitatea este deseori preferată de utilizatorul general, spre deosebire de securitate, este îndeajuns menționarea câtorva aplicații P2P (peer-to-peer) și VoIp (voice-over-ip). Deși unele au fost construite cu scopul

prioritizării securității (Freenet, PGP-phone, Nautilus [41–43]) acestea au eșuat în detrimentul unor aplicații ușor de folosit precum BitTorrent, Skype, dar care au adăugat încet caracterstici pentru securitatea lor.

În ultimii ani, se publică reportaje care încearcă să conștientizeze publicul larg despre importanța securității și intimității persoanelor, pentru o documentare mai bună înainte de a folosi anumite servicii [44, 45].

5.1 Arhitectură bazată pe scheme de partajare

Folosind cele 4 clase descrise mai sus, am ales să decuplăm securitatea de partea de design pentru ca pe viitor modificările aduse aplicației ca întreg să nu fie dificile de realizat. Acest lucru a fost atins prin crearea de API-uri (interfață pentru programarea aplicațiilor) cu ajutorul frameworkului Django REST (detalii în secțiunea **??**). Pentru partea de UI am folosit AngularJS cu scopul de a integra API-urile dezvoltate anterior [46].

Aducem la cunoștință că implementarea acestei arhitecturi este departe de a fi una ideală și nu trebuie folosită în producție, fiind vulnerabilă la atacuri (side-channel, bazate pe generatoare de numere pseudo-aleatoare, etc) [**?**, 47].

O privire de ansamblu a structurii proiectului se găsește în figura 5.1.

O dată ce utilizatorul se autentifică, acesta are acces la o listă cu toate fișierele partajate de el. Pentru a distribui un fișier pe instanțele EC2, alege numărul de mașini la care să-l împartă și dimensiunea pragului pentru care se poate face reconstituirea. La reconstituire are la dispoziție o listă cu toate fișierele partajate pe de la înregistrarea sa până în momentul actual.

5.2 Distribuția unui fișier

Fie $\mathcal{H}$ o funcție hash criptografic sigură [1]. La înregistrarea unei persoane, DealerApp generează în mod aleator $rpad \leftarrow^R \{0,1\}^{256}$ și îl păstrează privat.

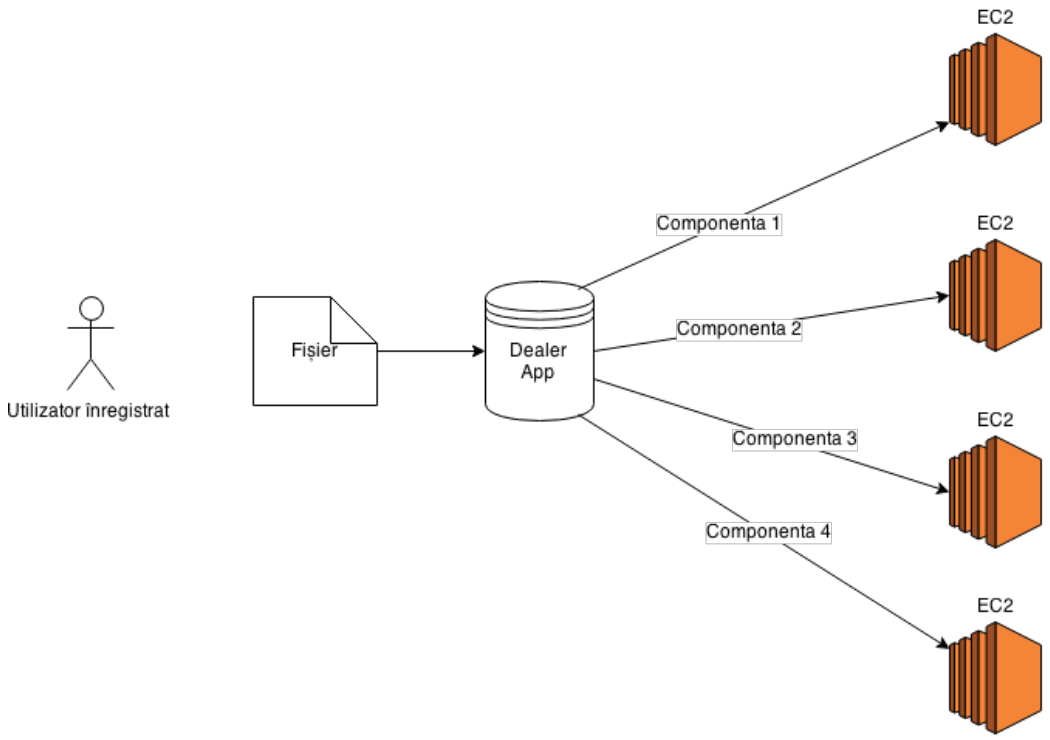

Figura 5.1: Distribuirea unui fișier în DealerApp la 4 instanțe EC2

Fișierul încărcat de utilizatorul autentificat este preluat de un API iar acesta este distribuit conform figurii 5.2. Codul sursă pentru schema Shamir a fost preluat de pe GitHub (sub licență Apache) și modificat potrivit necesităților [48].

De asemenea, modul în care erau generați coeficienții aleatori în [48] a fost modificat (din *random.randint* în *os.random*) din motive de securitate.

Documentația limbajului Python precizează că pentru a folosi un generator criptografic sigur este recomandată funcția *os.random* (apelează intern */dev/urandom*).

Studii recente arată că practic, cele mai sigure generatoare de numere pseudo-aleatoare sunt */dev/random* și */dev/urandom* din sistemele UNIX, acestea nu sunt robuste [49]. Principala dificultate fiind extragerea entropiei precum și măsurarea ei.

Date de intrare: Un fișier binar $\mathcal{S}$ având numele *name*;

Date de ieșire: n perechi de șiruri binare s_i, sgn_i;

Inițializare: n șiruri binare $s_i \leftarrow \emptyset$, $1 \leq i \leq n$;

Procesarea componentelor: Pentru fiecare octet b din $\mathcal{S}$, DealerApp:
- Generează un polinom f de grad $k-1$ având coeficienți aleatori;
- Termenul liber al polinomului devine b
  - Alege n valori aleator distincte $x_i \leftarrow^R GF(256) \setminus \{0\}$, $1 \leq i \leq n$;
  - Concatenează perechea $x_i, f(x_i)$ la sfârșitul șirului binar s_i, i.e. $s_i \leftarrow s_i||x_i||f(x_i)$;

Generarea semnăturilor: Pentru fiecare componentă s_i:

- $sgn_i \leftarrow \mathcal{H}(s_i||rpad||name)$

Distribuție: DealerApp:
- Distribuie șirul binar s_i împreună cu sgn_i instanței EC2 cu indicele i.

Figura 5.2: Schema DealerApp - Distribuire

5.3 Reconstrucția unui fișier

Utilizatorului autentificat i se pune la dispoziție o listă cu numele fișierelor partajate. Când utilizatorul alege unul dintre ele, DealerApp trimite cereri de componente la fiecare din instanțele EC2. Presupunând că fișierul a fost partajat astfel încât oricare k instanțe să fie capabile să îl reconstituie, DealerApp se va opri din a face cereri atunci când primește cel puțin k răspunsuri valide (Figura 5.3).

O componentă (s, h) provenită dintr-un fișier cu numele *name* este validă dacă $H(s||rpad||name) = h$.

În Figura 5.4 este prezentată refacerea unui document din 4 componente.

Pentru ca un adversar pasiv să nu obțină vreo informație despre ce se transmite pe firul de comunicație dintre utilizator (în special numele fișierelor dar și conținutul acestora) este necesar protocolul HTTPS.

HTTPS este protocolul standard de HTTP construit pe baza TLS [7] și necesită autorități certificate pentru generarea de chei private / publice. O dată stabilite aceste chei, datele transmise pot fi criptate. Cu toate acestea, am considerat că adoptarea protocolului HTTPS nu se află printre scopurile acestei lucrări.

Date de intrare: Cel puțin k componente valide (s_i, h_i) provenite din instanțe EC2.

Date de ieșire: Fișierul original $\mathcal{S}$;

Inițializare: Un șir binar $\mathcal{S} \leftarrow \emptyset$; L lungimea unui șir binar s_i; $L \leftarrow \frac{L}{2}$.

Reconstrucție: DealerApp pentru fiecare j, $1 \leq j \leq L$:

- Pentru fiecare șir binar din cele k: $xp_i \leftarrow x_{i_j}, yp_i \leftarrow y_{i_j}$ $1 \leq i \leq k$, unde cu x_{i_j}, y_{i_j} am notat cel de-al j-lea octet corespunzător pentru x din șirul i, respectiv y;
- Interpolează cele k puncte xp_i, yp_i și obține un octet b;
- Concatenează la sfârșitul lui $\mathcal{S}$ pe b i.e. $\mathcal{S} \leftarrow \mathcal{S}||b$

Figura 5.3: Schema DealerApp - Reconstrucție

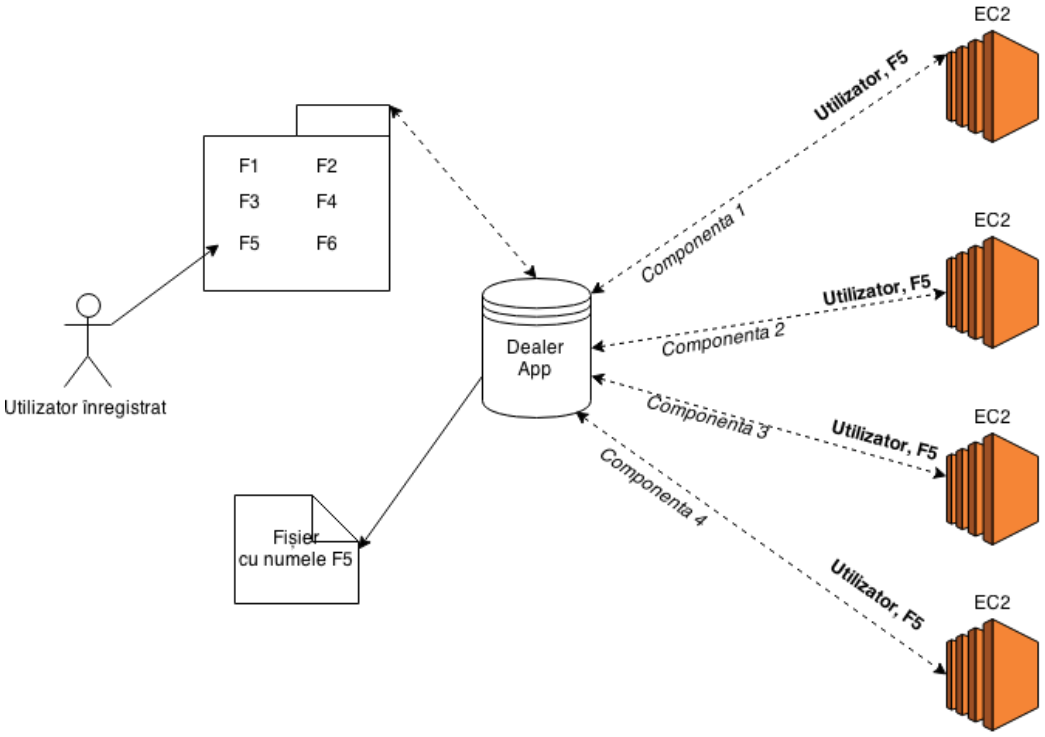

Figura 5.4: Reconstituirea unui fișier cu numele $F5$ în DealerApp din 4 instanțe EC2

5.4 Date reținute de DealerApp

5.5 Instanțe EC2

Amazon Web Services pun la dispoziție o gamă largă de servicii specializate pentru calculul în cloud. Acestă tehnică poartă denumirea de *grid computing* (asamblarea de resurse computaționale din locații multiple pentru un anumit scop) [50].

Printre uneltele AWS se numără și instanțele EC2, calculatoare cu configurații personalizate (tip de hard-disk, procesor CPU, volum de memorie RAM, procesor GPU, etc). Am considerat că ele reprezintă un început bun pentru crearea unei rețele proprii de calculatoare din motive precum:

- Numărul mare de clienți dar și companii precum Adobe Systems, NetFlix, Pinterest;
- Existența unor unelte puternic dezvoltate pentru programatori (Boto, consola AWS-cli);
- Posibilitatea de a încerca gratis, timp de 1 an, serviciile AWS (programul Free-Tier).

Conectarea la serviciile propriu-zise se face prin folosirea unei perechi de chei, ambele generate de AWS:

- Cheie bazată pe ID (Acces Key ID) - folosită pentru autentificarea cererilor și este unică pentru toți utlizatorii;
- Cheie secretă (Acces Secret Key) - folosită pentru pentru semnarea digitală fiecărei cereri.

AWS recomandă înlocuirea acestei perechi de chei o dată la 90 de zile pentru creșterea securității contului.

5.5.1 Stabilirea unui grup de securitate

AWS permite definirea unui grup (sau mai multe) de securitate prin care se specifică reguli ale traficului de rețea. O instanță clasică EC2 poate avea

asociate cel mult 500 de astfel de grupuri iar un grup este capabil de maxim 100 de reguli [51].

Regulile unui grup constă în blocarea sau admiterea anumitor porturi corespunzătoare protocoalelor TCP sau UDP dar și un număr de adrese IP care pot accesa mașinile asociate grupului [52, 53].

În particular, pentru implementarea arhitecturii DealerApp, poate fi aleasă o singură regulă de securitate și anume blocarea traficului de intrare/ieșire pe toate porturile și protocoale, mai puțin pe TCP cu portul 6379, cu care se realizează transmisia de date între server și baza de date Redis aflată pe instanțele EC2.

Bineînțeles, atunci când este necesară obținerea de log-uri de pe o anume instanță, se poate asocia incă un grup pentru efectua conexiunea prin SSH la mașina respectivă prin protocolul TCP având portul cu numărul 22 [54].

5.5.2 Imagini AMI

AMI (Imagini pentru Mașini Amazon) semnifică informația necesară pentru ca o instanță EC2 să poată porni. La prima creare a unei instanțe ($\mathcal{I}$) EC2 acestea sunt echipate cu un sistem de operare standard (Windows, Ubuntu, Redhat) la alegere, în cadrul programului Free-Tier [51].

După ce un utilizator a adăugat uneltele necesare pe $\mathcal{I}$, acesta are posibilitatea de a crea o poză sau snapshot instanței $\mathcal{I}$ care va salva starea într-o AMI. Prin stare înțelegem conținutul volumului rădăcină /, nu și starea proceselor din RAM.

O dată ce poza este finalizată, utilizatorul poate porni oricât de multe instanțe $\mathcal{J}$ care au aceeași stare cu $\mathcal{I}$ în momentul realizării pozei.

AMI sunt accesibile atât în consola AWS cât și prin intermediul API-urilor (exemplu Boto) pentru o manevrare mai ușoară a lor [55].

În cazul DealerApp, instanțele EC2 au fost pornite dintr-un AMI care conține un sistem de operare Ubuntu 14.04 și un server de Redis (Secțiunea 5.5.3).

Pentru pornirea automată a serverului Redis impreună cu instanța a fost necesară configurarea unui serviciu (Upstart) ce se ocupă cu startul, oprirea

și supervizarea aplicațiilor unei mașini. Upstart funcționează doar pe calculatoarele prezente cu un sistem de operare bazat pe Linux [56].

5.5.3 Redis

Redis (REmote DIctionary Server) este o bază de date *cheie-valoare* întemeiată pe asocierea unică unei chei cu o valoare [57]. Redis, spre deosebire celelalte baze de date *cheie-valoare*, poate asocia unei chei structuri de date complexe (nu doar șiruri de caractere) precum liste, mulțimi, mulțimi sortate, tabele de dispersie, etc [58].

Motivul pentru care am ales Redis este datorat simplității folosirii sale, rapiditatea cu care efectuează operațiile de citire, scriere dar și existența uneltor specializate pentru lucrul în limbajul Python [58, 59].

Conexiunile prin intermediul clientului Redis pentru Python sunt realizate prin conectarea la numele de host al serverului și numărul portului, fiind instanțiate de DealerApp.

În cazul intanțelor, la adresa TCP cu portul 6379 trebuie să se poate conecta doar adrese IP autorizate, spre exemplu adresa IP a clienților la care rulează DealerApp. În cazul acesta, instanța EC2 rămâne în continuare vulnerabilă la atacuri care implică falsificarea unor adrese IP. Pentru o securitate mai puternică este necesară semnarea interogărilor la baza de date, posibilă prin pornirea unei aplicații mai complexe (web-server) care să funcționeze cu același server Redis.

Din motive de simplitate, am renunțat la adăugarea unui alt strat de securitate pastrând doar serverul de Redis, profitând de avantajul că un adversar trebuie să compromită cel puțin $n-k+1$ instanțe pentru ca reconstituirea să nu fie posibilă, considerând că fișierul inițial a fost partajat folosind o schema (k, n) majoritară.

La primirea unei componente (sgn_i, s_i), instanța i salvează într-o tabelă hash cu cheia sgn_i valoarea s_i (folosind comanda Redis *hset* sgn_i s_i).

5.5.4 Evaluarea arhitecturii

Din punct de vedere al securității, aceasta este garantată dacă un adversar nu reușește să obțină cel puțin k componente la partajarea unui fișier. TODO dezvoltă

Integritatea componentelor este păstrată pentru că *rpad* rămăne privată. Dificultatea unui adversar constă în generarea de perechi (sgn_i, s_i) valide iar acest lucru este posibil doar dacă reușește să găsească o coliziune pentru funcția $\mathcal{H}$ (*rpad* este privat).

Găsirea unei coliziuni pentru o funcție hash criptografic sigure necesită resurse computaționale extraordinar de mari iar atacul nu este fezabil pentru un adversar PPT [1]. În particular, pentru funcția de hash SHA3-256 cele mai eficiente atacuri sunt bazate pe paradoxul zilei de naștere și au o complexitate de aproximativ 2^{128} operații [60].

Corectitudinea protocolului este asigurată din schema Shamir și faptul că la reconstituire se folosesc doar componente care au fost semnate inițial de utilizator cu ajutorul funcției SHA3 [61].

Concluzii

Referințe

[1] Katz, J., Lindell, Y.: Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series). Chapman & Hall/CRC (2007)

[2] Storer, M.W., Greenan, K.M., Miller, E.L., Voruganti, K.: Potshards - a secure, recoverable, long-term archival storage system. TOS **5**(2) (2009)

[3] R.F.Olimid, D.A.Rotaru: An immediate multi-party generalization of ID-NIKE from constrained PRF. In: Analele Universității București. Theory Day in Computer Science (2014)

[4] R.F.Olimid, D.A.Rotaru: On the security of a backup technique for database systems based on threshold sharing. Journal of Control Engineering and Applied Informatics (în recenzie) (2015)

[5] 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

[6] Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. Journal of the ACM (JACM) **33**(4) (1986) 792–807

[7] Dierks, T.: The transport layer security (tls) protocol version 1.2. (2008)

[8] Daemen, J., Rijmen, V.: The design of Rijndael: AES-the advanced encryption standard. Springer Science & Business Media (2013)

[9] Boneh, D., Waters, B.: Constrained pseudorandom functions and their applications. In: Advances in Cryptology-ASIACRYPT 2013. Springer (2013) 280–300

[10] Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. Journal of the ACM (JACM) **56**(6) (2009) 34

[11] Diffie, W., Hellman, M.E.: New directions in cryptography. Information Theory, IEEE Transactions on **22**(6) (1976) 644–654

[12] Joux, A.: A one round protocol for tripartite diffie–hellman. In: Algorithmic number theory. Springer (2000) 385–393

[13] Boneh, D., Silverberg, A.: Applications of multilinear forms to cryptography. Contemporary Mathematics **324**(1) (2003) 71–90

[14] Shamir, A.: Identity-based cryptosystems and signature schemes. In: Advances in cryptology, Springer (1985) 47–53

[15] Paterson, K.G., Srinivasan, S.: On the relations between non-interactive key distribution, identity-based encryption and trapdoor discrete log groups. Designs, Codes and Cryptography **52**(2) (2009) 219–241

[16] Boneh, D.: The decision Diffie-Hellman problem. In: Algorithmic number theory. Springer (1998) 48–63

[17] Blakley, G.: Safeguarding cryptographic keys. Proceedings of the 1979 AFIPS National Computer Conference (1979) 313–317

[18] Shamir, A.: How to share a secret. Commun. ACM **22**(11) (1979) 612–613

[19] 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

[20] 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

[21] 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

[22] Chor, B., Goldwasser, S., Micali, S., Awerbuch, B.: Verifiable secret sharing and achieving simultaneity in the presence of faults. In: 2013 IEEE 54th Annual Symposium on Foundations of Computer Science, IEEE (1985) 383–395

[23] 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

[24] 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

[25] Beimel, A.: Secret-sharing schemes: a survey. In: Coding and cryptology. Springer (2011) 11–46

[26] 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

[27] Subbiah, A., Blough, D.M.: An approach for fault tolerant and secure data storage in collaborative work environments. In: StorageSS. (2005) 84–93

[28] 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

[29] 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

[30] Storer, M.W., Greenan, K., Miller, E.L.: Long-term threats to secure archives. In: Proceedings of the second ACM workshop on Storage security and survivability, ACM (2006) 9–16

[31] 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

[32] 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

[33] 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

[34] WebSite: Python Programming Language - official website (2015) Ultima accesare: Iunie, 2015, https://www.python.org/.

[35] Hunter, J.D.: Matplotlib: A 2D graphics environment. Computing In Science & Engineering **9**(3) (2007) 90–95

[36] WebSite: Cerealizer package (2015) Ultima accesare: Iunie, 2015, https://pypi.python.org/pypi/Cerealizer.

[37] WebSite: Github - Official Website (2015) Ultima accesare: Iunie, 2015, https://www.github.com.

[38] WebSite: Cod Github - Official Website (2015) Ultima accesare: Iunie, 2015, https://www.github.com/rdragos/splitting-scheme.

[39] WebSite: Europass - download examples (2014) Ultima accesare: Iunie, 2015, http://www.europass.cedefop.europa.eu/ro/home.

[40] Gutmann, P., Grigg, I.: Security usability. Security & Privacy, IEEE **3**(4) (2005) 56–58

[41] WebSite: Freenet project (2015) Ultima accesare: Iunie, 2015, https://freenetproject.org/.

[42] WebSite: Pgpfone - pretty good privacy phone (2015) Ultima accesare: Iunie, 2015, http://www.pgpi.org/products/pgpfone/.

[43] WebSite: Nautilus secure phone (2015) Ultima accesare: Iunie, 2015, http://www.lst.de/~johannes/nautilus-securephone/.

[44] Greenwald, G., MacAskill, E.: Nsa prism program taps in to user data of apple, google and others. The Guardian **7**(6) (2013) 1–43

[45] Greenwald, G., MacAskill, E., Poitras, L.: Edward snowden: the whistleblower behind the nsa surveillance revelations. The Guardian **9** (2013) 2013

[46] WebSite: Angular website (2015) Ultima accesare: Iunie, 2015, https://angularjs.org/.

[47] Kocher, P.C.: Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In: Advances in Cryptology—CRYPTO'96, Springer (1996) 104–113

[48] WebSite: Python shamir secret sharing scheme (2015) Ultima accesare: Iunie, 2015, `https://github.com/hbs/PySSSS`.

[49] Dodis, Y., Pointcheval, D., Ruhault, S., Vergniaud, D., Wichs, D.: Security analysis of pseudo-random number generators with input:/dev/random is not robust. In: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, ACM (2013) 647–658

[50] Garfinkel, S.L.: An evaluation of amazon's grid computing services: Ec2, s3, and sqs. In: Center for, Citeseer (2007)

[51] WebSite: Amazon user guid website (2015) Ultima accesare: Iunie, 2015, `http://docs.aws.amazon.com/AWSEC2/latest/UserGuide`.

[52] Protocol, U.D.: Rfc 768 j. postel isi 28 august 1980. Isi (1980)

[53] Forouzan, B.A.: TCP/IP protocol suite. McGraw-Hill, Inc. (2002)

[54] Ylonen, T., Lonvick, C.: The secure shell (ssh) protocol architecture. (2006)

[55] WebSite: boto: A python interface to amazon web services (2015) Ultima accesare: Iunie, 2015, `http://boto.readthedocs.org/en/latest/`.

[56] WebSite: Upstart (2015) Ultima accesare: Iunie, 2015, `http://upstart.ubuntu.com/`.

[57] Cattell, R.: Scalable sql and nosql data stores. ACM SIGMOD Record **39**(4) (2011) 12–27

[58] WebSite: Redis data types (2015) Ultima accesare: Iunie, 2015, `http://redis.io/documentation`.

[59] WebSite: Client redis pentru python (2015) Ultima accesare: Iunie, 2015, `https://pypi.python.org/pypi/redis/`.

[60] Dinur, I., Dunkelman, O., Shamir, A.: Collision attacks on up to 5 rounds of sha-3 using generalized internal differentials. In: Fast Software Encryption, Springer (2014) 219–240

[61] Bertoni, G., Daemen, J., Peeters, M., Van Assche, G.: Keccak sponge function family main document. Submission to NIST (Round 2) **3** (2009) 30