UNIVERSITATEA ”ALEXANDRU IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

Căutare în date criptate

propusă de

Antonia-Luciana Bursuc

Sesiunea: *iunie, 2017*

Coordonator științific

Lect. Dr. Sorin Iftene

UNIVERSITATEA ”ALEXANDRU IOAN CUZA” DIN IAȘI

FACULTATEA DE INFORMATICĂ

Căutare în date criptate

Antonia-Luciana Bursuc

Sesiunea: *iunie, 2017*

Coordonator științific

Lect. Dr. Sorin Iftene

DECLARAŢIE PRIVIND ORIGINALITATE ŞI RESPECTAREA DREPTURILOR DE AUTOR

Prin prezenta declar că Lucrarea de licenţă cu titlul *Căutarea în date criptate* este scrisă de mine şi nu a mai fost prezentată niciodată la o altă facultate sau instituţie de învăţământ superior din ţară sau din străinătate. De asemenea, declar că toate sursele utilizate, inclusiv cele preluate de pe Internet, sunt indicate în lucrare, cu respectarea regulilor de evitare a plagiatului:

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

Iaşi, *27.06.2017* Absolvent *Antonia-Luciana Bursuc*

_____

DECLARAŢIE DE CONSIMŢĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul *Căutare în date criptate*, codul sursă al programelor şi celelalte conţinuturi (grafice, multimedia, date de test etc.) care însoţesc această lucrare să fie utilizate în cadrul Facultăţii de Informatică.

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

Iaşi, *27.06.2017* Absolvent *Antonia-Luciana Bursuc*

_____

Cuprins

Introducere

O dată cu răspândirea la scara larga a internetului, tot mai multe companii au ales să-și păstreze datele în cloud. Stocarea unui număr uriaș de date, ușor accesibile, la cerere, clienților, pe diverse tipuri de dispozitive și costurile scăzute au fost câteva din avantajele evidente care au determinat creșterea popularității serverelor la distanță. Riscurile se manifestă însă la nivelul securității și apare problema prevenției accesului neautorizat la resursele stocate.

Atacurile asupra acestor date sunt o amenințare permanentă și diversă. Pericolele își pot avea originea atât în exterior, prin intruși pasivi sau activi, care exploatând vulnerabilitățile sistemului sau sfidând măsurile de precauție, reușesc să descopere sau să modifice informații sensibile, cât și în interior, prin administratori de baze de date curioși sau rău-intenționați care pot avea cu ușurință acces la date confidențiale. Prevenirea tuturor acestor forme de scurgere a datelor s-a dovedit a fi o reală provocare, numeroase companii înregistrând anual pierderi semnificative. Diverse metode de protecția precum verificări la nivelul sistemului de operare sau a rețelei, analiza codului sau hardwardul de încredere, contribuie la securizarea împotriva atacurilor din afara sistemului, dar nu garantează niciun fel de protecția împotriva unui intrus din interior.

Soluția ideală în această situație devine criptarea datelor astfel încât doar clientul autorizat să poată avea acces la datele în variantă lizibilă, iar serverul de baze de date să pătreze doar varianta criptată, oferită de acesta.

În acest punct, își face simțită prezența o altă dificultate: ce procedură se aplică scenariului, uzual de altfel în lumea reală, în care clientul păstrează pe server un număr

mare de date și dorește să preia doar acele documente care conțin un anumit termen. Un răspuns trivial în această situație este: serverul trimite toate datele clientului, care le decriptează și realizează căutarea local. Bineînțeles că astfel stocarea în cloud ar fi irelevantă, din moment ce clientul ar putea păstra toate datele local, deținând spațiu de stocare și putere de calcul suficiente, iar căutarea ar fi astfel mai rapidă, scutită fiind de întâzierile de timp datorate transferului. Totuși clientul ar putea să nu dețină suficient spațiu de stocare și atunci ar fi nevoie să primească datele pe rând, cauzând un deficit de performanță și mai crescut. O astfel de abordare nu este fezabilă în situații reale. O soluție convenabilă în situații practice este criptarea care permite căutări (engl: *Searcheable Encryption*). Acest tip de schemă de criptare păstrează datele private prin criptare, dar permite căutarea de termeni în criptotext, fără ca serverul care realizează operația să obțină informații cu privire la termenul căutat sau la plaintext.

Totuși, chiar fiind posibilă filtrarea documentelor după un cuvânt furnizat de client, mai există o altă problema la fel de presantă. În cazul unei baze de date criptate, cum se pot efectua interogări peste anumite intervale (engl: *range queries*) sau cum s-ar putea ordona datele? De asemenea, dacă baza de date este de mari dimensiuni, o situație frecventă de altfel, cum s-ar putea construi indecși care să înlesnească căutarea și să reducă timpul de procesare a unei interogări? Un răspuns la această provocare a fost oferit prin introducerea ideii de schemă de *criptare care păstrează ordinea* (sau OPE, engl: *Order Preserving Encryption*). Într-o astfel de schemă, relația de ordine dintre plaintexte rămâne aceeași și pentru criptotexte.

Lucrarea de față este structurată în patru capitole. În primul capitol, am prezentat instrumentele fundamentale pe care se bazează protocoalele criptografice, atât la nivel teoretic, cât și prin exemple de construcție. Cel de-al doilea capitol tratează scenariul în care, dată fiind o colecție de documente criptate, se dorește preluarea acelora dintre ele care conțin un anumit cuvânt. În cel de-al treilea capitol, am selectat cele mai importante protocoale ce ilustrează traseul pe care l-a urmat criptografia ca urmare a necesității, tot mai presante, de a stoca și opera în mod eficient cu date numerice criptate. În ultimul capitol am prezentat modul în care am implementat câte unul din protocoalele discutate în capitolele al doilea și al treilea.

Contribuții

Principalele contribuții personale sunt următoarele:

- consultarea literaturii de specialitate și citirea de articole relevante pentru tema aleasă
- selectarea celor mai importante articole
- prezentarea celor mai relevante tehnici de căutare în date criptate într-o manieră unitară
- implementarea câte unui protocol pentru fiecare din cele două scenarii avute în vedere: tehnici de criptare care permit căutarea unui cuvânt în criptotext și tehnici de criptare care păstrează ordinea.

1. Primitive criptografice

Primitivele criptografice sunt instrumente criptografice de bază utilizate în construcția de protocoale. În această secțiune vom trece în revistă cele mai importante primitive folosite în construcțiile descrise în capitolele următoare. Informațiile au fost preluate, în special, din [1], [2] și [3].

1.1. Generator de numere pseudo-aleatoare

Un generator pseudo-aleator de numere denumit și PRNG (engl: *Pseudo Random Number Generator*) sau PRG (engl: *Pseudo Random Generator*) este o funcție g de complexitate timp poliomială care transformă un șir de dimensiuni reduse, denumit și sămânță sau *seed*, într-un șir de dimensiuni întinse, $g(seed)$ care apare ca fiind aleator pentru orice adversar.

Un adversar este un algoritm care încearcă să facă distincție între șirul astfel generat și un șir aleator de aceeași dimensiune. Cele două șiruri sunt asemenea dacă probabilitatea de apariție pentru fiecare dintre cele două șiruri este aceeași.

Definiția 1: *Un generator de numere pseudo-aleatoare* (t, ε) *este o funcție* $G : \{0,1\}^n \to \{0,1\}^m$ *cu* $m \gg n$ *dacă:*

- *G poate fi calculat în mod eficient de un algoritm determinist ;*

- *Pentru orice algoritm adversar A cu timp de rulare t:*

$$|Pr[A(G(S))|S \leftarrow \{0,1\}^n] - Pr[A(R)|R \leftarrow \{0,1\}^m]| < \varepsilon$$

Cu alte cuvinte, distribuția uniformă peste mulțimea $\{0,1\}^m$ este (t,ε)-indistinctibilă față de distribuția $\{G(S)|S \leftarrow \{0,1\}^n\}$.

Un generator pseudo-aleator convertește printr-un proces determinist eficient o secvență aleatoare scurtă într-o secvență aleatoare de dimensiune mai amplă și de aceea, în practică entropia șirului de ieșire nu depășește pe aceea a șirului de intrare.

1.2. Funcții pseudo-aleatoare

Conceptul de funcție pseudo-aleatoare sau PRF (engl: *Pseudo-Random Function*) a fost introdus de Goldreich, Goldwasser și Micali în [4] și denumește o funcție care este indistinctibilă față de o funcție selectată din mulțimea tuturor funcțiilor definite peste același domeniu și codomeniu.

Definiția 2: *O funcție $f : \{0,1\}^n \times \{0,1\}^s \to \{0,1\}^m$ este o funcție pseudo-aleatoare (t,ε,q) dacă:*

- *dată fiind o cheie $K \in \{0,1\}^s$ și o valoare $X \in \{0,1\}^n$ există un algoritm polinomial care calculează $F_K(X) = F(K,X)$;*
- *pentru orice algoritm oracol de timp t, este adevărată relația*:

$$|Pr_{K \leftarrow \{0,1\}^s}[A^{f_K}] - Pr_{f \in \mathcal{F}}[A^f]| < \varepsilon,$$

unde $\mathcal{F} = \{0,1\}^n \to \{0,1\}^m$, iar A face cel mult q interogări.

1.3. Permutări pseudo-aleatoare

Permutările pseudo-aleatoare, pe scurt PRP (engl: *Pseudo-Random Permutations*) au fost introduse de Luby și Rackoff în [1] și formalizează bine cunoscuta noțiune de criptare bloc. Schemele de criptare bloc sunt scheme cu cheie privată care criptează fiecare bloc de plaintext printr-un bloc de criptotext de aceeași dimensiune. De aceea, se poate considera că această criptare induce de fapt o *permutare* asupra șirului de caractere inițial.

Astfel, un *sistem ideal de criptare bloc* poate fi definit ca o colecție $E = \{\pi_1, \pi_2, \ldots \pi_{2^s}\}$ de permutări *aleatoare* $\pi_i : \{0,1\}^n \to \{0,1\}^n$, iar criptarea este descrisă prin mulțimea $C = \pi_k(M)$, unde k este cheia aparținând mulțimii $\{\pi_1, \ldots, \pi_{2^s}\}$, iar decriptare este $M = \pi^{-1}(C)$. Cum permutările cu adevărat aleatoare nu există, în practică sunt folosite permutările *pseudo-aleatoare*.

Luby și Rackoff au definit securitatea permutărilor pseudo-aleatoare în funcție de diferitele atacuri considerate în contextul criptării bloc și au propus următoarele definiții:

- permutările sigure împotriva unui atac cu plaintext ales, adică un adversar eficient care are acces la criptotextele plaintextelor alese, nu poate distinge, cu o probabilitate neneglijabilă, între acestea și niște permutări aleatoare.

  Definiția 3: $\pi : \{0,1\}^s \times \{0,1\}^n \to \{0,1\}^n$ *este o permutare pseudo-aleatoare* (t, ε, q) *dacă:*

  - *Pentru orice* $K : \{0,1\}^s$, π_K *este o funcție injectivă de la* $\{0,1\}^n$ *la* $\{0,1\}^n$ *;*
  - *Pentru orice* $K : \{0,1\}^s$, *există un algoritm polinomial pentru a evalua* $\pi_K(X)$*;*
  - *Pentru orice algoritm A cu timp de rulare t este îndeplinită relația*

    $$|Pr_{K \leftarrow \{0,1\}^s}[A^{\pi_K}] - Pr_{\sigma \leftarrow \mathcal{P}}[A^{\sigma}]| < \varepsilon,$$

    unde A face q interogări și $\mathcal{P} = \{\pi : \{0,1\}^n \to \{0,1\}^n, \pi = permutare\}$ *este mulțimea tuturor permutărilor peste* $\{0,1\}^n$.

- permutările pseudo-aleatoare ”puternice” sau SPRP (engl: *Strong PRP*) sunt sigure împotriva unui atac adaptativ cu plaintext ales și criptotext ales.

  Definiția 4: $\pi : \{0,1\}^s \times \{0,1\}^n \to \{0,1\}^n$ *este o permutare pseudo-aleatoare puternică* (t, ε, q) *dacă:*

  - *Pentru orice* $K : \{0,1\}^s$, π_K *este o funcție injectivă de la* $\{0,1\}^n$ *la* $\{0,1\}^n$;
  - *Pentru orice* $K : \{0,1\}^s$, *există un algoritm polinomial pentru a evalua* $\pi_K(X)$ *și* $\pi_K^{-1}(X)$;
  - *Pentru orice algoritm A cu timp de rulare t este îndeplinită relația*

    $$|Pr_{K \leftarrow \{0,1\}^s}[A^{\pi_K, \pi_K^{-1}}] - Pr_{\sigma \leftarrow \mathcal{P}}[A^{\sigma, \sigma^{-1}}]| < \varepsilon,$$

    unde A face q interogări.

1.4. Funcții Hash

Termenul de funcție hash se referă la o funcție care transformă un șir de dimensiune variabilă într-unul de dimensiune fixată sau mai formal:

Definiția 5: *O funcție hash este o funcție* $h : D \to R$ *unde domeniul* $D = \{0,1\}^*$ *și* $R = \{0,1\}^n$ *pentru* $n \geq 1$.

Funcțiile hash criptografice trebuie însă să respecte câteva cerințe suplimentare și au un mare număr de aplicații în criptografie precum: autentificare, semnături digitale, generare de numere pseudo-aleatoare, time stamping, etc.

Funcțiile hash criptografice pot fi :

- cu cheie
  - sunt numite și coduri de autentificare a mesajelor
- fără cheie

- sunt asociate deobicei termenului de funcție hash
- sunt numite și coduri de detectare a manipulării (sau MDC, engl: *Manipulation Detection Code*)
- în concordanță cu propritetățile adiționale, pot fi subclasificate în:
  * funcții hash one-way
  * funcții hash rezistente la coliziune

O funcție hash one-way respectă următoarele cerințe:

1. să poată fi aplicată unui bloc de date de dimensiune oricât de mare
2. returnează un rezultat de dimesiune fixă
3. date fiind funcția H și valoarea de intrare x, $H(x)$ este ușor de calculat
4. dată fiind valoare $H(x)$ este imfesabil de a afla valoarea lui x
5. dată fiind funcția H și valoarea $H(x)$ este infesabil de a găsi x și x' astfel încât $H(x) = H(x')$.

Funcție hash rezistentă la coliziune este o funcție care satisface toate cerințele unei funcții hash one-way împreună cu următoarea proprietate:

dată fiind funcția H atunci este infesabil de a găsi o pereche (x, y), $x \neq y$ astfel încât H(x) = H(y).

În continuare vor fi prezentate două exemple de construcție de funcții hash. Informațiile au fost preluate din [5] și [6].

MD5 În 1991, Richard Rivest a propus o versiune îmbunătățită a funcției *md4*, iar în 1992, aceasta a fost publicată în [6] sub denumirea de *MD5*.

Fie mesajul m de n biți lungime,

$$m = m_0 m_1 \dots m_{n-1}.$$

Pentru a calcula valoare funcției hash $MD5(m)$, se vor parcurge următorii pași:

1. Mesajul va fi extins astfel încât lungimea lui să fie congruentă cu 448 (mod 512), adică cu 64 de biți mai puțin decât un multiplu de 512. Aceasta se va realiza adăugând un bit "1", urmat de oricâți de "0" este nevoie. Extinderea se face chiar dacă lungimea este deja congruentă cu 448 (mod 512). Numărul minim de biți adăugați va fi 1, iar numărul maxim 512.

2. O reprezentare pe 64 de biți a lui n (lungimea inițailă a mesajului) este concatenată la rezultatul obținut la pasul anterior.

   Dacă $n > 2^{64}$, ceea ce este puțin probabil, se vor utiliza primii 64 cei mai nesemnificativi biți. În acest punct, lungimea mesajului obținut este multiplu de 512 biți. Mesajul va fi împărțit în blocuri, iar lungimea fiecărui bloc este multiplu de 16. Fie t numărul de blocuri astfel obținute.

3. Se vor utiliza 4 blocuri A, B, C, D de 32 de biți, ce vor fi inițializate cu valorile hexadecimale ale următoarelor variabile:

   $$h_0 \leftarrow 67452301$$
   $$h_1 \leftarrow \text{efcdab89}$$
   $$h_2 \leftarrow \text{98badcfe}$$
   $$h_3 \leftarrow 10325476$$

4. Fie următoarele funcții:
   $f(j,x,y,z) = (x \wedge y) \vee (\neg(x) \wedge z) \qquad (0 \leq j \leq 15)$
   $f(j,x,y,z) = (x \wedge z) \vee (y \wedge \neg(z)) \qquad (16 \leq j \leq 31)$
   $f(j,x,y,z) = x \oplus y \oplus z \qquad (32 \leq j \leq 47)$
   $f(j,x,y,z) = (x \vee \neg(z)) \oplus y \qquad (48 \leq j \leq 63)$

   Fiecare funcție definită are la intrare 3 blocuri de 32 de biți, iar la ieșire un bloc de 32 de biți. Prima funcție acționează ca o condiționare de forma: dacă x atunci y altfel z. Se folosește, de asemenea, un tabel de 64 de elemente $K[1 \ldots 64]$ construit cu ajutorul funcției sinus.

   $K(j) =$ primii 32 biți din $sin(j+1) \quad (0 <= j <= 63)$

Se procesează fiecare bloc multiplu de 16 și se copie fiecare bloc j în r:

$r(j) = j \qquad (0 \leq j \leq 15)$

$r(j) = (1 + 5 * (j - 16)) \pmod{16} \qquad (16 \leq j \leq 31)$

$r(j) = (5 + 3 * (j - 32)) \pmod{16} \qquad (32 \leq j \leq 47)$

$r(j) = 7 * (j - 48) \pmod{16} \qquad (48 \leq j \leq 63)$

Matricea s conține numărul de poziții pentru fiecare rotație ce urmează a fi efectuată în fiecare din cele 4 runde:

$s(0 \ldots 15) = 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22, 7, 12, 17, 22$

$s(16 \ldots 31) = 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20, 5, 9, 14, 20$

$s(32 \ldots 47) = 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23, 4, 11, 16, 23$

$s(48 \ldots 63) = 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21, 6, 10, 15, 21$

Urmează 4 runde de procesare, fiecare rundă având câte 16 pași. Astfel, pentru fiecare bloc vor fi efectuați în total 64 de pași de procesare. Pseudocodul funcției $MD5$ este următorul:

$A \leftarrow h_0 \; ; \quad B \leftarrow h_1$

$C \leftarrow h_2 \; ; \quad D \leftarrow h_3$

for $i \leftarrow 0$ to $t - 1$ do

  for $j \leftarrow 0$ to 63 do

    $TEMP \leftarrow A + f(j, B, C, D) + X[i][r(j)] + K(j);$

    $A \leftarrow D \; ; \qquad D \leftarrow C \; ; \qquad C \leftarrow B$

    $B \leftarrow B + rol(TEMP, s(j))$

  end

  $h_0 \leftarrow h_0 + A \; ; \quad h_1 \leftarrow h_1 + B$

  $h_2 \leftarrow h_2 + C \; ; \quad h_3 \leftarrow h_3 + D$

end

5. Se returnează h_0, h_1, h_2, h_3.

SHA La 31 ianuarie 1992, Institutul Național de Standarde și Tehnologie (engl: *National Institute for Standards and Technology*) a publicat în Registrul Federal (engl: *Federal Register*) un standard de securitate hash care conține o descriere a *Secure Hash Algorithm*, pe scurt SHA și denumit ulterior SHA-1.

Algoritmul se prezintă ca o versiune îmbunătățită a *MD4*. Mărimea blocurilor, utilizate în calcul, este de 512 de biți. Rezultat este pe 160 de biți, ceea ce face din *sha* o funcție mai puțin vulnerabilă la un atac de tip *brute force*. Numărul de pași din fiecare rundă este 20, iar numărul de runde este 4. Modalitatea de extindere a mesajului primit la intrare este aceași ca în cazul *MD*5.

Algoritmul utilizează o secvență de funcții logice $f(0), f(1), \ldots, f(79)$, fiecare având la intrare 3 blocuri de câte 32 de biți, iar la ieșire un bloc de 32 de biți. Pentru $0 \leq t \leq 79$ și cuvintele B, C, D, funcțiile se definesc după cum urmează:

$$
\begin{aligned}
f(t,B,C,D) &= (B \wedge C) \vee ((\neg B) \wedge D) && (0 \leq t \leq 19) \\
f(t,B,C,D) &= B \oplus C \oplus D && (20 \leq t \leq 39) \\
f(t,B,C,D) &= (B \wedge C) \vee (B \wedge D) \vee (C \wedge D) && (40 <= t <= 59) \\
f(t,B,C,D) &= B \oplus C \oplus D && (60 \leq t \leq 79).
\end{aligned}
$$

Se va utiliza și o secvență de constante hexadecimale $K(0), K(1), \ldots, K(79)$, definite astfel:

$$
\begin{aligned}
K(t) &= \text{5a827999} && (0 \leq t \leq 19) \\
K(t) &= \text{6ed9eba1} && (20 \leq t \leq 39) \\
K(t) &= \text{8f1bbcdc} && (40 \leq t \leq 59) \\
K(t) &= \text{ca62c1d6} && (60 \leq t \leq 79).
\end{aligned}
$$

Se inițializează valorile hexadecimale:

$$
\begin{aligned}
h_0 &\leftarrow \text{67452301} \\
h_1 &\leftarrow \text{efcdab89}
\end{aligned}
$$

$h_2 \leftarrow$ 98badcfe

$h_3 \leftarrow$ 10325476

$h_4 \leftarrow$ c3d2e1f0.

Pseudocodul funcției este:

$A \leftarrow h_0$; $B \leftarrow h_1$

$C \leftarrow h_2$; $D \leftarrow h_3$

$E \leftarrow h_4$

for $i \leftarrow 0$ to $t-1$ do

  for $j \leftarrow 0$ to 79 do

    if $(j > 15)$ then

      $X[i][j] \leftarrow X[i][j-3] \oplus X[i][j-8] \oplus X[i][j-14] \oplus X[i][j-16]$

    end

    $TEMP \leftarrow rol(A,5) + f(j,B,C,D) + E + X[i][j] + K(j)$

    $E \leftarrow D$

    $D \leftarrow C$

    $C \leftarrow rol(B,30)$

    $B \leftarrow A$

    $A \leftarrow TEMP$

  end

$h_0 \leftarrow h_0 + A$

$h_1 \leftarrow h_1 + B$

$h_2 \leftarrow h_2 + C$

$h_3 \leftarrow h_3 + D$

$h_4 \leftarrow h_4 + E$

end

1.5. Criptare simetrică

În cadrul sistemelor de criptare simetrică, denumite și sisteme de criptare cu cheie privată, există o singură cheie secretă folosită atât la criptare, cât și la decriptare. Într-un astfel de sistem dacă o entitate A dorește să transmită un mesaj m, criptat ca c, unei entități B, atunci aceasta va trebui să se asigure că B cunoaște cheia folosită pentru criptarea mesajului. Astfel, cea mai importantă problemă, denumită și *problema de distribuire a cheii* (engl: *key distribution problem*) se referă la găsirea unei modalități eficiente de a transmite această cheie lui B.

Există două tipuri de criptare simetrică:

- criptare bloc (engl: *block cipher*): schemă de criptare în care plaintextul este împărțit în blocuri de dimensiune egală, iar criptarea se realizează pentru fiecare bloc în parte. În cazul în care plaintextul nu are dimensiunea adecvată, se va realiza o extindere (engl: *padding*) a ultimului bloc.
- criptare secvențială (engl: *stream cipher*): criptare bloc, în care dimensiunea blocului este 1. Se folosește în cazul în care erorile de transmisie au o probabilitate mare de apariție deoarece erorile nu se propagă.

AES Algoritmul de criptare bloc *Rijndael*, creat de Daemen și Rijmen, a fost adoptat începând cu anul 2001 ca standard de criptare sub denumirea de *AES* (engl: *Advanced Encryption Standard*).

Lungimea unui bloc este de 128 de biți, iar lungimea cheii poate fi de 128, 192 sau 256 de biți. Numărul de runde de procesare depinde de lungimea cheii: pentru cheia de 128 de biți se execută 10 runde, pentru cea de 192 de biți, 12 runde, iar pentru cea de 256 de biți, 14 runde. Fie nr, numărul de runde. AES folosește $nr+1$ chei de rundă (engl: *roundkey*) pe 128 de biți, obținute din cheia inițială.

Algoritmul pentru criptarea unui mesaj m, având 128 de biți, spre a obține criptotextul c, urmează pașii:

1. Se inițializează matricea de stare $state \leftarrow m$

2. Se efectuează un sau exclusiv între cheia de rundă și matricea de stare $roundkey \oplus state$

3. (a) Pentru fiecare rundă de la 1 la $nr-1$, $state$ va fi procesat urmând pașii:

   i. Efectuează o substituție de octeți, folosind o $S-cutie$,

   ii. Efectuează o deplasare a liniilor

   iii. Efectuează o combinare a coloanelor

   (b) Pentru cea de-a nr-a rundă efectuează $roundkey \oplus state$

4. (a) Efectuează o subtituție de octeți, folosind o $S-cutie$

   (b) Efectuează o deplasare a liniilor

   (c) Efectuează $roundkey \oplus state$

5. Criptotextul $c \leftarrow state$

Variabila $state$ este o matrice 4×4, care va păstra în fiecare celulă câte două cifre hexadecimale.

O $S-cutie$ este o matrice de mărie 16×16 cu liniile și coloanele notate cu câte o cifră hexadecimală. Astfel, elementul de pe linia x și coloana y este accesat prin $S-cutie(x,y)$. Matricea conține în fiecare celulă câte două cifre hexadecimale. Substituția octeților presupune ca fiecare octet din matricea $state$, notat xy, să fie înlocuit cu elementul $S-cutie(x,y)$.

Deplasarea liniilor matricei de stare presupune câte o deplasare a fiecărei linii în parte, adică o rotație spre stânga cu $x-1$, unde x este indexul liniei.

Combinarea coloanelor presupune înmulțirea între o matrice dată de forma 4×4 și fiecare coloană y din matricea de stare pentru a obține câte o nouă coloană:

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \cdot \begin{bmatrix} stare_{0,y} \\ stare_{1,y} \\ stare_{2,y} \\ stare_{3,y} \end{bmatrix} = \begin{bmatrix} stare'_{0,y} \\ stare'_{1,y} \\ stare'_{2,y} \\ stare'_{3,y} \end{bmatrix}$$

1.6. Criptare Asimetrică

În sistemele de criptare cu cheie publică, denumite și sisteme de criptare asimetrică, fiecare entitate A deține o cheie publică e și o cheie privată d. Un astfel de sistem este sigur dacă fiind dat e, este infesabil de a calcula d.

Orice entitate B, ce dorește să trimită un mesaj m către A, va obține cheia publică e a lui A, prin criptare va obține $c = Enc(e,m)$, iar apoi va trimite c lui A. Pentru a-l putea citi, acesta va decripta mesajul folosind cheia privată: $m = Dec(d,c)$. Această tehnică elimină dificultatea distribuirii cheilor secrete din cazul criptării simetrice. Principalul obiectiv este menținerea confidențialității datelor trimise între părți.

Din moment ce algoritmul de criptare este public, criptarea asimetrică nu asigură autenticitatea originii datelor sau verificarea integrității. Acestea vor fi asigurate prin mesaje de autentificare și semnături digitale.

O astfel de schemă este în general mai lentă decât una simetrică. Din acest motiv, acestă tehnică este utilizată, cel mai adesea, pentru transferul de chei sau criptarea unor mesaje de dimensiuni reduse, precum numărul unui card.

RSA Criptosistemul RSA, numit astfel după inventatorii săi, Rivest, Shamir și Adleman, este cel mai utilizat criptosistem asimetric. Este utilizat în asigurarea menținerii secrete a datelor și pentru semnături digitale.

Criptosistemul RSA poate fi descris astfel:

- *KeyGen*

1. Generează două numere prime mari p și q, unde $p \neq q$;
2. Calculează $n = pq$ și $\phi(n) = (p-1)(q-1)$;
3. Generează aleator un e, unde $1 < e < \phi(n)$, astfel încât $cmmdc(e, \phi(n)) = 1$;
4. Utilizează Euclid extins pentru a calcula $d, 1 < d < \phi(n)$, astfel încât

$$ed \equiv 1 \pmod{\phi(n)};$$

ieșire: cheia publică (n, e) și cheia privată d.

- *Enc*
  intrare: cheia publică (n, e) și mesajul m
  1. reprezintă mesajul m ca un număr întreg din intervalul $[0, \ldots, n-1]$ ($m \in \mathbb{Z}_n$)
  2. Calculează $c = m^e \pmod{n}$

  ieșire: criptotextul c

- *Dec*
  intrare: cheia privată d și m

  Calculează $m = c^d \pmod{n}$

  ieșire: mesajul m

2. Tehnici de căutare a unui cuvânt într-un document criptat

2.1. Tehnici bazate pe criptare simetrică

În această secțiune vom prezenta trei scheme de criptare simetrică care permit căutări de cuvinte în criptotext (sau SSE, engl: *Searchable Symmetric Encryption*).

Prima dintre acestea a fost propusă de Song, Wagner și Perrig în [8]. Aceasta are la bază 3 primitive:

- G – un generator de secvențe pseudo-aleatoare,
- F – o funcție pseudo-aleatoare,
- E – o permutare pseudo-aleatoare.

Pentru mai multe informații a se vedea capitolul 1.

Căutarea în text se realizează în mod secvențial.

Pentru a cripta un document ce conține o secvență de cuvinte W_1, W_2,...,W_l vor fi generate l secvențe pseudo-aleatoare $S_1, S_2, \ldots, S_l$ de $n-m$ biți utlizând generatorul G.

Criptarea propriu-zisă a unui cuvânt W_i de dimensiune n se realizează aplicând operația de XOR peste W_i și termenul T_i care a fost obținut prin concatenarea S_i cu $F_{k_i}(S_i)$.

Așadar, $C_i := W_i \oplus T_i$, unde $T_i := \langle S_i, F_{k_i}(S_i) \rangle$. Astfel, doar cel care a generat secvențele $T_1, T_2, \ldots, T_l$ are posibilitatea de decriptare.

Se observă faptul că fiecare cuvânt poate fi criptat independent de celelalte. La fiecare pas (adică pentru fiecare cuvânt) din document există posibilitatea de a alege dacă

cheia k_i folosită să coincidă cu una din cheile folosite anterior sau să fie diferită. Altfel spus, pentru fiecare poziție i din text, $k_i = k_j$ sau $k_i \neq k_j$, unde $j < i$.

Pentru decriptare, proprietarul documentului va trimite la server cuvântul căutat W și cheia k_i, unde i este locația în care cuvântul W poate să apară. Serverul poate căuta cuvântul în text, într-un interval de timp liniar, verificând dacă $C_i \oplus W_i$ este de forma $\langle s, F_{k_i}(s) \rangle$ pentru un s oarecare.

Există însă inconveniente serioase în această abordare: proprietarul trebuie să cunoască pozițiile în care poate apărea cuvântul W sau altfel dacă, dorește căutarea unui cuvânt în tot documentul, va trebui să dezvălui serverului toate cheile și deci conținutul întregului document. Aceste probleme pot fi adresate introducânt o nouă funcție pseudo-aleatoare.

Fie $f : K_F \times \{0,1\}^* \to K_F$, o altă funcție pseudo-aleatoare. Această funcție va avea o cheie secretă k', aleasă inițial în mod uniform aleator din mulțimea K. Mai apoi cheile k_i pentru fiecare cuvânt din text vor fi $k_i := f_{k'}(W_i)$.

Pentru căutare unui cuvânt în text, proprietarul va dezvălui serverului $f_{k'}(W)$ și W. Astfel se realizează controlul căutării, întrucât serverul nu poate afla decât locația cuvântului transmis, independent de alte cuvinte din text.

Această schemă nu suportă căutări ascunse ale unui termen deoarece pentru a realiza căutarea unui cuvânt W, proprietarul va dezvălui serverului acest cuvânt, W, însă pentru a remedia aceste aspect se va introduce o nouă etapă de criptare a textului, precedentă criptării descrise mai sus.

În această nouă etapa de pre-criptare, fiecare cuvânt din text va fi criptat separat folosind un algoritm determinist de criptare $E_{k''}$. Astfel secvența de cuvinte $W_1, W_2, ...W_l$ va deveni $X_1, X_2, \ldots, X_l$, unde $X_i := E_{k''}(W_i)$. Criptarea unui cuvânt nu depinde de poziția pe care se află, ci doar de valoare lui. Algoritmul de criptare E poate fi văzut ca un algoritm de criptare bloc ECB sau CBC cu un vector de inițializare IV, în cazul unor cuvinte suficient de lungi, care se aplică în mod similar pentru fiecare termen W_i din document. În etapa de criptare propriu-zisă, se efectuează XOR peste fiecare bloc de criptotext X_i corespunzător unui cuvânt și termenul T_i obținut prin concatenarea dintre X_i și $F_k(X_i)$.

Pentru a permite serverului să realizeze o căutare a uni cuvânt W în documentul criptat, proprietarul trimite serverului perechea $\langle X, k \rangle$, unde X corespunde criptării

cuvântului dorit cu algoritmul determinist E, iar k este obținut prin apelul funcției f folosind cheia secretă k' și valoarea de intrare X. Atfel spus $X := E_{k''}(W_i)$, iar $k := f_k(X)$. Se observă că atâta timp cât algoritmul de criptare E este sigur, proprietatea de interogare ascunsă este satisfăcută întrucât serverul are acces la cuvântul căutat, doar în variantă criptată.

Se observă însă că schema prezentată anterior are un mare inconvenient și anume: dacă $k_i := f_k(E_{k''}(W_i))$ atunci nici proprietarul textului nu va fi capabil să decripteze documentul întrucât nu cunoaște $E_{k''}(W_i)$, mai precis primii m biți din $E_{k''}(W_i)$.

Fiecare secvență X_i obținută în etapa de pre-criptare va fi împărțită în L_i, cuprinzând primii $n-m$ biți și R_i, cuprinzând ultimii m biți, adică $X_i := \langle L_i, R_i \rangle$. Cheia k_i va fi obținută calculând $f_{k'}(L_i)$.

Pentru decriptare, proprietarul documentului va putea genera S_i pentru fiecare poziție i din text întrucât deține sămânța (*seed*) de generare a secvenței pseudo-aleatoare și va efectua operația de XOR cu primii $n-m$ biți din C_i, obținând L_i, adică primii $n-m$ biți din cuvântul criptat cu algoritmul determinist E. Având L_i, se va calcula cheia $k_i := f_{k'}(L_i)$ și va continua decriptarea realizând XOR cu ultimii m biți din C_i, ceea ce reprezintă ultimii m biți din $E_{k''}(W_i)$, adică R_i. În acest moment, acesta deține X_i, criptat în mod simetric cu algoritmul E și cheia k'', pe care îl va aplica din nou, iar rezultatul este cuvântul W_i.

În cazul în care cuvântul W_i nu trece prin faza de pre-criptare, schema nu prezintă siguranță deoarece este foarte probabil ca mai multe cuvinte dint text să aibă primii $n-m$ biți identici. Acestă etapă este astfel neccesară pentru ca valorile lui L_i, pentru fiecare i din text să fie distincte. Dată fiind cheia k_i serverului, acesta nu va afla nimic mai mult decât poziția la care cuvântul a fost găsit.

Așadar, presupunând că E este o permutarea pseudo-aleatoare (t, l, e_F)- sigură, F o funcție pseudo-aleatoare (t, l, e_f)-sigură si G un generator pseudo-aleator (t, e_G)- sigur, iar fiecare cheie $k_i = f_{k_i}(L_i)$ si $L_i = \rho_x(E_{k''}(W_i))$. Atunci algoritmul H, care generează secvențele $\langle T_1, T_2, ..., T_l \rangle$ va fi un generator pseudo-aleator $(t - \varepsilon, e_H)$- sigur.

Următoarele două scheme au fost propuse de Reza Curtmola și alții în [9].

Fie $\Delta = \{W_1, W_2, \ldots\}$ un dicționar de cuvinte. Se asociază un index I cu o colecție de documente D. Indexul este compus din două structuri de date:

- un vector A care conține în formă criptată o mulțime $D(W)$ pentru fiecare cuvânt $W \in \Delta$;
- un tabel dicționar T care conține informația care permite localizarea și decriptarea elementelor corespunzătoare din A pentru fiecare cuvânt $W \in \Delta$.

Există o colecție de liste înlănțuite $L_i, W_i \in \Delta$, unde nodurile fiecărei liste L_i sunt indentificatorii documentelor în $D(W_i)$. În vectorul A sunt înscrise nodurile tuturor listelor L_i, în ordine aleatoare și criptate cu ajutorul unei chei generate în mod aleator. Înainte de criptare, la cel de-al j-lea nod al listei L_i se concatenează informația celui de-al $(j+1)$-lea nod din L_i, împreună cu cheia ce a fost folosită la criptarea lui.

În acest mod, fiind dată poziția (index-ul) din A și cheia de decriptare pentru primul nod al listei L_i, serverul va fi capabil să localizeze și să decripteze toate nodurile listei L_i. Astfel, dată fiind o listă L_i, mărimea acesteia nu va fi cunoscută întrucât nodurilor tututor listelor L_i sunt stocate în vectorul A într-o ordine arbitrară.

Pentru construirea tabelului de căutare care permite localizarea și decriptare primelor elemente ale fiecărei liste L_i, fiecare intrare din T corespunde unui cuvânt $W_i \in \Delta$ și conține o pereche de forma $< adresă, valoare >$:

- Câmpul *valoare* conține poziția din A și cheia de decriptare pentru primul element al listei L_i. Acest câmp este criptat prin intermediul valorii returnate de o funcție pseudo-aleatoare.
- Câmpul *adresă* este pur și simplu folosit pentru a localiza o intrare în T. Tabelul este organizat prin adresare indirectă.

Utilizatorul va calcula atât A cât și T utilizând D în clar și le va stoca pe server împreună cu varianta criptată a lui D. Atunci când utilizatorul dorește să recupereze documentele care conțin cuvântul W_i, va determina cheia de decriptare și adresa pentru intrarea corespunzătoare din T și le va trimite către server.

Serverul localizează și decriptează intrarea dată din T și ia poziția din A și cheia de decriptare pentru primul element din L_i. Din moment ce fiecare element din L_i conține informația despre următorul element al listei L_i, serverul poate localiza și decripta toate nodurile din L_i, care oferă identificatorii din $D(W_i)$.

Intrările din tabelul T sunt tuple $< adresă, valoare >$, în care câmpul *adresă* este folosit ca o adresă virtuală pentru a localiza intrarea din tabelul T care conține valoarea din câmpul *valoare* corespunzător. Fiind dat un parametru p, adresa virtuală face parte dintr-un domeniul de dimensiune exponențială, adică $\{0,1\}^p$, iar numărul maxim de intrări în tabel va fi polinomial în p.

Dacă un tabel T, are câmpul adresă din $\{0,1\}^p$, câmpul *valoare* din $\{0,1\}^v$ și există cel mult m intrări în T, atunci se va spune că T este un dicționar de forma $(\{0,1\}^p \times \{0,1\}^v \times m)$.

Fie $Addr$ mulțimea de adrese virtuale folosite pentru intrările din tabelul T. Se poate stoca în mod eficient T prin organizarea mulțimii $Addr$ într-un dicționar FKS, o structură de date eficientă pentru stocarea tabelelor rare care necesită $O(|Addr|)(+o(|Addr|))$ memorie și $O(1)$ timp de căutare. Așadar, dată fiind o anumită adresă virtuală A, există două posibilități:

- $A \in Addr$: se va returna valoarea corespunzătoare, într-un timp constant;
- $A \notin Addr$: adresa A nu este definită.

Fie k, l parametri de securitate, iar (K, Enc, Dec) o schemă de criptare semantic sigură cu $Enc : \{0,1\}^l \times \{0,1\}^r \to \{0,1\}^r$. Fie funcția pseudo-aleatoare (a se vedea secțiune 1.2.) f și permutările pseudo-aleatoare π și ψ (a se vedea secțiune 1.3.) având parametrii:

- $f : \{0,1\}^k \times \{0,1\}^p \to \{0,1\}^{l+log2(m)}$
- $\pi : \{0,1\}^k \times \{0,1\}^p \to \{0,1\}^p$
- $\psi : \{0,1\}^k \times \{0,1\}^{log_2(m)} \to \{0,1\}^{log_2(m)}$.

Fie m mărimea totală a colecției de documente exprimată în unități, unde o unitate reprezintă dimensiunea minimă posibilă a unui cuvânt (de exemplu, un bit). Fie A

un vector de dimensiune m. Fie T un dicționar de forma $(\{0,1\}^p \times \{0,1\}^{l+log_2(m)} \times |\Delta|)$, administrat prin acces indirect, în felul mai sus specificat.

Construcția schemei are loc după cum urmează:

KeyGen(1^k, 1^l): generează în mod aleator cheile $s, y, z \leftarrow \{0,1\}^k$ și returnează $K = (s, y, z, 1^l)$

BuildIndex(K, D):

1. Initializare:
   a) scanează D și construiește Δ', mulțimea de cuvinte distincte din D. Pentru fiecare cuvânt $W \in \Delta'$, construiește $D(W)$;
   b) inițializează contorul global $ctr = 1$.
2. Construiește vectorul A:
   a) pentru fiecare $W_i \in \Delta'$:(construiește o listă L_i și memoreaz-o în vectorul A)
      - generează $k_{i,0} \leftarrow \{0,1\}^l$
      - pentru fiecare $1 \leq j \leq D(W_i)$:
        - generează $k_{i,j} \leftarrow \{0,1\}^l$ și nodul $N_{i,j} = \langle id(D_{i,j}) || k_{i,j} || \psi_s(ctr+1)$, unde $id(D_{i,j})\rangle$ este al j-lea identificator din $D(W_i)$;
        - calculează $\varepsilon_{k_{i,j-1}}(N_{i,j})$ și se memorează $A[\psi_s(ctr)]$
        - $ctr = ctr + 1$
      - pentru ultimul nod din L_i $(N_i, |D(W_i)|)$ înainte de criptare, se asignează adresa următorului nod la $NULL$
   b) fie $m' = \sum_{W_i \in \Delta'} |D(W_i)|$. Dacă $m' < m$ atunci setează pentru restul $(m - m')$ intrărilor din A valori aleatoare de aceași mărime ca cele m' intrări existente din A
3. Construiește tabelul T:
   a) pentru fiecare $W_i \in \Delta$:
      - $valoare = \langle addr(A(N_{i,1})) || k_{i,0} \rangle \oplus f_y(W_i)$

- setează $T[\pi_z(W_i)] = valoare$

b) dacă $|\Delta'| < |\Delta|$, atunci oferă celorlalte $(|\Delta| - |\Delta'|)$ intrări din T valori aleatoare.

4. Ieșire $I = (A, T)$.

Trapdoor(W): returnează $T_W = (\pi_z(W), f_y(W))$.

Search(I, T_W):

1. Fie $(\gamma, \eta) = T_W$, $\theta = T[\gamma]$, $\langle \alpha, k \rangle = \theta \oplus \eta$
2. Decriptează lista L începând cu nodul de la adresa α, criptat cu cheia k.
3. Întoarce lista cu identificatorii documentului conținuți de L.

În ceea ce privește eficiența, fiecare interogare necesită o singură rundă de căutare, iar mesajul are mărimea $O(1)$.

Memoria necesară:

- utilizatorului: $O(1)$;
- serverului: $O(m)$ (serverul stochează baza de date D criptată de dimensiune $O(m)$ și indexul I care conține: vetorul A de mărime $O(m)$ și dicționarul T de dimensiune $O(|\delta|)$).

Timpul necesar:

- utilizatorului pentru a calcula un cuvânt-trapă: $O(1)$;
- serverului pentru procesarea unei interogări pentru W: proporțional cu $|D(W)|$.

În ceea ce privește securitatea schemei, aceasta dezvăluie:

- rezultatul unei căutări
- patern-ul căutării
- mărimea totală a colecției de documente

- numărul de documente din D

  Fie $m' = \sum_{W_i \in \Delta_i} |L_i|$.

- Dacă pentru orice document D_j din colecția de documente D, un cuvânt nu are mai mult de o apariție în D_j, atunci $m = m'$.

- Dacă mărimea lui A este mai mică decât m, atunci reiese faptul că măcar un document D conține un cuvânt de mai multe ori, iar pentru a evita această scurgere de informație, se va seta mărimea lui A ca fiind egală cu m, completând restul de $(m - m')$ intrări cu valori aleatoare.

În mod similar, pentru a evita dezvăluirea numărului de cuvinte diferite din D (T are o intrare pentru fiecare cuvânt diferit din D), se adaugă $(|\Delta| - |\Delta'|)$ intrări aleatoare în T. Astfel numărul de intrări din T este mereu $|\Delta|$.

Având în vedere cele de mai sus, schema de față este o schemă non-adaptativ sigură, adică prezintă siguranță doar față de un atacator care nu ține cont de rezultatele căutărilor anterioare. Cum acest scenariu nu este în concordanță cu realitatea, se vor aduce câteva modificări și se ajunge la cea de-a doua versiune a schemei. Această nouă versiune prezintă avantajul securității înpotriva unui adversar adaptativ.

Pentru un cuvânt dat W și un întreg j, se va realiza concatenarea $W||j$. *Familia* cuvântului $W \in \Delta$ se definește ca o mulțime de etichete $F_W = \{W||j : 1 \leq j \leq D(W)\}$. Astfel, pentru fiecare cuvânt $W \in \Delta$ se vor întroduce în index elementele familiei F_W a cuvântului, în locul $D(W)$, așa cum a fost prezentat în schema anterioară.

Căutarea unui cuvânt devine echivalentă cu o căutare a acestor elemente. Cum fiecare etichetă din familia cuvântului va apărea doar într-un singur document, o căutare a cuvântului va dezvălui doar o locație în index.

Se va asocia colecției de documente D un index I care constă într-un tabel pentru căutări T. Pentru fiecare etichetă din familia unui cuvânt, se va adăuga o intrare în T, a cărei câmp valoare este identificatorul documentului care conține o instanță a cuvântului W.

Pentru a ascunde numărul de cuvinte din document, se vor adăuga elemente tabelului T astfel încât identificatorul fiecărui document să apară în același număr de

intrări. Pentru căutarea unui cuvânt, utilizatorul va fi nevoit să caute prin toate etichetele familiei acestuia.

Fie k un parametru de securitate. Se va utiliza funcția pseudo-aleatoare $\pi : \{0,1\}^k \times \{0,1\}^p \rightarrow \{0,1\}^p$. Fie $m = max * n$, unde n este numărul de documente din colecția de documente D, iar max dimensiunea celui mai mare document din D. Fie T un tabel de forma $(\{0,1\}^p \times \{0,1\}^{log_2(n)} \times m)$, în care se utilizează un sistem de adresare indirectă.

Construcția schemei are loc după cum urmează:

KeyGen(1^k, 1^l): generează în mod aleator cheia $s \leftarrow \{0,1\}^k$ și returnează $K = s$

BuildIndex(K, D):

1. Inițializare:
   - scanează D și construiește Δ, mulțimea cuvintelor distincte din D. Pentru fiecare cuvând $W \in \Delta$, construiește $D(W)$.
2. Construiește tabelul T:
   a) pentru fiecare $W_i \in \Delta$:
      - pentru $1 \leq j \leq |D(W_i)|$:
        * valoare = id($D_{i,j}$), unde id($D_{i,j}$) este al $j - lea$ identificator din $D(W_i)$;
        * setează T[$\pi(W_i||j)$] = valoare

   b) fie $m' = \sum_{W_i \in D(W_i)} |D(W_i)|$. Dacă $m' < m$, atunci setează celelate $(m - m')$ intrări astfel încât pentru fiecare $D' \in D$, valoare = id(D') pentru exact max intrări și setează câmpul adresă al acestor intrări cu valoari aleatoare.

   c) Ieșire: $I = T$.
3. Trapdoor: returnează $T_W = (T_{W_1}, \ldots, T_{W_{max}}) = (\pi_s(W||1), \ldots, \pi_s(W||max))$.
4. Search(I, T_W): Pentru $1 \leq i \leq max$: obține id=T[T_{W_i}] și returnează id.

Din punct de vedere al securității această schemă este *adaptativ* sigură, adică prezintă siguranță conform următoarei definiții:

Definiția 1: *O schemă simetrică de criptare care permite căutări este sigură în sensul adaptabilității dacă pentru oricare două istoricuri de căutare construite în mod adaptativ, de dimensiune egală, niciun adversar nu poate, într-un timp polinomial, distinge între cele două cu o probabilitate neneglijabilă mai mare ca 1/2 .*

Pentru fiecare interogare sunt necesare o rundă de comunicație între client și server și spațiu de stocare și timp de calcul proporționale cu numărul de documente care vor fi furnizate ca răspuns. Utilizatorul va avea nevoie de timp de calcul și spațiu de stocare de ordinul $O(1)$. Timpul de comunicare este egal cu *max* și spațiul de stocare suplimentar (față de prima schema) este $O(max)$.

2.2. Tehnici bazate pe criptare asimetrică

Pentru a rezolva problema căutărilor în date criptate în cazul în care se folosește criptografie cu cheie publică (a se vedea secțiunea 1.5.), Dan Boneh în colaborare cu Rafail Ostrovsky, Giuseppe Persianoz, Giovanni Di Crescenzo au elaborat în [10], un protocol care are la bază criptarea bazată pe identitate (sau IBE, *Identity Base Encryption*), așa cum va fi denumită în continuare.

Necesitatea unei criptări cu cheie publică care permite căutarea unui cuvânt (sau PEKS, *Public-key Encryption with Keyword Search*) răspunde unui scenariu în care proprietarul cheii publice primește email-uri criptate cu această cheie și dorește să le sorteze în funcție de niște cuvinte-cheie. Pentru a face aceasta proprietarul ar dori să poată căuta termeni în criptotext fără ca server-ului de email să i se dezvăluie conținutul mesajului sau termenul căutat.

Schema de criptare ce permite căutări se definește prin intermediul a 4 algoritmi probabiliști polinomiali:

1. *KeyGen*

   intrare: parametru de securitate s

   ieșire: cheia publică A_{pub} și cheia privată A_{priv}

2. *PEKS*

   intrare: cheia publică A_{pub}, cuvântul W

   ieșire: o criptare a cuvântului W ce permite căutări

3. *Trapdoor*

   intrare: cheia publică A_{pub}, cuvântul W

   ieșire: un cuvânt-trapă T_W pentru cuvântul dat W

4. *Test*

   intrare: cheia publică A_{pub}, o criptare care permite căutări $S = PEKS(A_{pub}, W')$ și un cuvânt-trapă $T_W = Trapdoor(A_{pub}, W)$

$$\textit{ieșire}: \begin{cases} \text{"}yes\text{"}, & \text{dacă } W = W' \\ \text{"}no\text{"}, & \text{altfel} \end{cases}$$

Securitatea împotriva unui atacator activ este definită prin următorul joc de securitate:

1. Provocatorul rulează algoritmul $KeyGen(s)$ pentru a genera A_{pub}, A_{priv} și-i oferă cheia publică atacatorului.

2. Atacatorul cere în mod adaptativ provocatorului cuvântul-trapă T_W pentru orice cuvânt $W \in \{0,1\}^*$ ales.

3. La un moment dat, atacatorul A alege două cuvinte W_0, W_1 și i le trimite provocatorului. Singura restricție este ca aceste două cuvinte să nu se numere printre cele pentru care atacatorul a interogat provocatorul până în momentul respectiv. Provocatorul alege un bit $B \in \{0,1\}$ și-i trimite atacatorului provocarea $C = PEKS(A_{pub}, W_b)$.

4. Atacatorul continuă să aleagă cuvinte W și să ceară cuvinte-trapă pentru acestea atâta timp cât $W \neq W_0, W_1$.

5. În final, atacatorul întoarce un bit $b' \in \{0,1\}$ și câștigă provocarea dacă $b = b'$.

Avantajul pe care îl are atacatorul A în a câștiga provocarea este definit ca $Adv(s) = |Pr[b = b'] - 1/2|$.

Sistemul este numit sigur împotriva unui atac de cuvânt-cheie ales dacă pentru orice atacator ce rulează în timp polinomial, $Adv(s)$ este o funcție neglijabilă.

Prima variantă de construcție a protocolului se bazează pe problema de calcul Diffie-Hellman. Pentru construirea sistemului IBE s-a folosit o aplicație biliniară pe curbe eliptice.

Mai precis, sunt folosite două grupuri G_1, G_2 de ordin prim p și o funcție de formă biliniară între cele două, $e : G_1 \times G_1 \to G_2$. Aceasta satisface următoarele proprietăți:

1. *Calculabilă*: fiind date $g, h \in G_1$ există un algoritm care poate calcula în timp polinomial $e(g,h) \in G_2$;

2. *Biliniară*: pentru orice întregi x și y avem $e(g^x, g^y) = e(g,g)^{xy}$;

3. *Non-degenerativă*: dacă g este generator al lui G_1, atunci $e(g^x, g^y)$ este generator al lui G_2.

Mărimea celor două grupuri este determinată de mărimea parametrului de securitate.

Construcția schemei de căutare pornește de la această formă biliniară și folosește funcțiile hash $H_1 : \{0,1\}^* \to G_1$ și $H_2 : G_2 \to \{0,1\}^{logp}$ (a se vedea secțiunea 1.4.). Modalitatea de funcționare a sistemului de criptare ce permite căutări este următoarea:

- *KeyGen*

  intrare: parametru de securitate

  Acesta va determina marimea, p, a grupurilor G_1 și G_2. Algoritmul generează aleator $\alpha \in Z_p^*$ și un generator g al lui G_1.

  ieșire: $A_{pub} = [g, h = g^\alpha]$ și $A_{priv} = \alpha$

- *PEKS*

  intrare: A_{pub}, W

  Se generează aleator $r \in Z_p^*$ și se calculează $t = e(H_1(W), h^r) \in G_2$

  ieșire: $[g^r, H_2(t)]$

- *Trapdoor*

  intrare: A_{priv}, W

  ieșire: $T_W = H_1(W)^\alpha \in G_1$

- *Test*

  intrare: $A_{pub}, S, T_{\mathrm{W}}$, cu $S = [A, B]$

  ieșire: $\begin{cases} \text{"}yes\text{"}, & \text{dacă } H_2(e(T_{\mathrm{W}}, A)) = B \\ \text{"}no\text{"}, & \text{altfel} \end{cases}$

Într-adevăr ,

$$\begin{aligned} H_2(e(T_W, A)) &= H_2(e(H_1(W)^{\alpha}, g^r)) \\ &= H_2(e(H_1(W), (g^r)^{\alpha})) \\ &= H_2(e(H_1(W), (g^{\alpha})^r)) \\ &= H_2(e(H_1(W), h^r)) \\ &= H_2(t) \\ &= B \end{aligned}$$

Acest sistem este o schema de criptare care permite căutari, non-interactivă și semantic sigură la un atac de cuvânt-cheie ales în modelul oracolului aleator.

Demonstrația securității se bazează pe varianta computațională a problemei Diffie-Hellman pentru aplicații biliniare.

Diffie-Hellman Bilianiar (BDH)

Fie g generatorul grupului G_1. Problema BDH se definește astfel:

intrare: $g, g^a, g^b, g^c \in G_1$

ieșire: $e(g,g)^{abc} \in G_2$

Spunem că BDH este intractabilă dacă orice algoritm de complexitate polinomială are un avantaj neglijabil de a rezolva problema.

Sistemul de criptare bazată pe indentitate propus de Boneh-Franklin este întemeiat pe aceiași presupunere pentru securitate. Securitatea sistemului propus reiese din următoarea teoremă:

Teorema 1 *Schema de criptare ce permite căutări descrisă mai sus este semantic sigură împotriva unui atac de cuvânt-cheie ales în modelul random oracle atâta timp cât problema BDH este intractabilă.*

Demonstrație:

Presupunem că A este un algoritm de atac care are avantajul ε în a compromite schema de criptare propusă. Să presupunem că A face cel mult

- q_{H_2} interogări către funcția H_2,
- q_{H_T} interogări pentru cuvinte-trapă.

Vom construi algoritmul B care rezolvă problema BDH cu probabilitate de cel puțin $\varepsilon' = \varepsilon/(e \cdot q_T \cdot q_{H_2})$, unde e este baza logaritmului natural. Timpul de rulare al algoritmului B este aproximativ același cu cel al lui A.

Așadar dacă presupunerea cu privire la problema BDH este adevărată pentru G_1, atunci ε' este o funcție neglijabilă și, în consecință, ε trebuie să fie o funcție neglijabilă pentru parametrul de securitate.

Fie g generatorul grupului G_1. Algoritmul B este definit astfel:

intrare: $g, u_1 = g^{\alpha}, u_2 = g^{\beta}, u_3 = g^{\gamma}$

ieșire: $v = e(g,g)^{\alpha\beta\gamma}$

Algoritmul B interacționează cu atacatorul A după cum urmează:

KeyGen. Algoritmul A începe prin a-i da lui B cheia publică $A_{pub} = [g, u_1]$

Interogări - H_1, H_2. În orice moment atacatorul A poate interoga oracolele aleatoare H_1 sau H_2.

Pentru a răspunde interogărilor către H_1, B menține o listă cu tuple de forma $\langle W_{\mathrm{i}}, h_{\mathrm{i}}, a_{\mathrm{i}}, c_{\mathrm{i}} \rangle$ denumită lista-H_1. Inițial acesta este goală. Atunci când A interoghează oracolul H_1 pentru un element $W_{\mathrm{i}} \in \{0,1\}^*$, algoritmul B răspunde după cum urmează:

1. Dacă W_{i} apare deja într-un tuplu al lista-H_1 atunci B răspunde cu $H_1(W_{\mathrm{i}} = h_i \in G_1)$
2. Altfel, B generează aleator un $c_i \in \{0,1\}$ astfel încât Pr[$c_i = 0$] = $1/(q_T + 1)$
3. Algoritmul B generează aleator $a_i \in Z_p$

   B calculează $\begin{cases} h_i \leftarrow u_2 \cdot g^{a_i} \in G_1, & \text{dacă } c_i = 0 \\ h_i \leftarrow g^{a_i} \in G_1, & \text{dacă } c_i = 1 \end{cases}$
4. Algoritmul B adaugă $\langle W_i, h_i, a_i, c_i \rangle$ în lista-H_1 și răspunde lui A, setând $H_1(W_i) = h_i$.

De asemenea, algoritmul B ține evidența interogărilor către H_2 prin intermediul listei lista-H_2 de tuple de forma (t,V). Inițial, acesta este goală. În momentul în care A lansează o interogare $H_2(t)$, algoritmul B alege o valoare $V \in \{0,1\}^{logp}$ pentru fiecare element nou t și fixează $H_2(t) = V$, adăungând noul tuplu în lita-H_2.

Interogări cuvinte-trapă. Atunci când A face o interogare pentru a obține trapa corespunzătoare unui cuvânt W_i, algoritmul B răspunde după cum urmează:

1. Algoritmul B apelează algoritmul definit anterior pentru a obține $h_i \in G_1$ asfel încât $H_1(W_i) = h_i$. Fie $\langle W_i, h_i, a_i, c_i \rangle$ tuplul corespunzător din lista-H_1.

2. Dacă $c_i = 0$ atunci B semnalează eșecul și își încheie execuția.

3. Dacă $c_i = 1$ atunci știm că $h_i = g^{a_i} \in G_1$. Se definește $T_i = u_1^{a_i}$. Deoarece $T_i = H_1(W_i)^{\alpha}$, acesta este cuvântul-trapă corespunzător cuvântului W_i. Algoritmul B returnează T_i.

Provocarea. La un moment dat algoritmul A alege o pereche de cuvinte-cheie W_0 și W_1 pentru care dorește să fie provocat. Algoritmul B generează provocarea PEKS astfel:

1. Algoritmul B apelează de două ori algoritmul care răspunde interogrilor-H_1 pentru a obține $h_0, h_1 \in G_1$ astfel încât $H_0(W_i) = h_0$ și $H_1(W_i) = h_1$. Pentru $i = 0,1$, fie $\langle W_i, h_i, a_i, c_i \rangle$ tuplele corespunzătoare din lista-H_1. Dacă $c_0 = 1$ și $c_1 = 1$ atunci B semnalează eșecul și își încheie execuția.

2. Altfel, cel puțin unul dintre c_0, c_1 este 0 și atunci B alege un bit aleator $b \in \{0,1\}$ astfel încât $c_b = 0$. Dacă doar unul dintre c_b este 0 atunci nu este necesară alegerea aleatoare, neavând de fapt decât o singură opțiune.

3. Algoritmul B răspunde cu provocarea PEKS $C = [u_3, J]$, unde J este generat aleator din $\{0,1\}^{logp}$

   Observație: Implicit, $H_2(e(H_1(W_b), u_1^{\gamma})) = J$. Astfel,

$$J = H_2(e(H_1(W_b), u_1^{\gamma})) = H_2(e(u_2 g^{a_b}, g^{\alpha\gamma})) = H_2(e(g^{\beta} g^{a_b}, g^{\alpha\gamma})) = H_2(e(g, g^{\alpha\gamma(\beta + a_b)}))$$

   Așadar, C este o informație PEKS validă pentru cuvântul W_b.

Mai multe interogări de cuvinte-trapă. A continuă să facă interogări pentru a obține trapele unor cuvinte W_i cu restricția $W_i \neq W_0, W_1$. Algoritmul B răspunde acestor interogări ca și mai înainte.

Ieșirea. La un moment dat, A va returna bitul $b' \in \{0,1\}$, adică provocarea C este rezultatul $PEKS(A_{pub}, W_0)$ sau $PEKS(A_{pub}, W_1)$. În acest moment algoritmul B alege aleator o pereche (t,V) pentru lista-H_2 și returnează $t/e(u_1,u_3)^{a_b}$ ca valoare pentru $e(g,g)^{\alpha\beta\gamma}$, unde a_b este valoarea folosită la setarea provocării.

Pentru a demonstra că acest sistem funcționează precum a fost prezentat în ipoteză, iar B rezolvă problema BDH cu probabilitate de cel puțin $\varepsilon' = \varepsilon/(eq_{\mathrm{T}}q_{\mathrm{H}_2})$, se ține cont de probabilitatea de apariție a 3 evenimente E_1, E_2, E_3, prezentate în următoarele 3 propoziții, împreună cu probabilitățile aferente și argumentările acestor rezultate.

Propoziția 1: Probabilitatea ca B să nu abandoneze execuția ca urmare a unei interogări a lui A pentru un cuvânt-trapă este cel puțin $1/e$.Așadar $Pr[E_2] \geq 1/e$.

Demonstrație: Având în vedere că s-a presupus că atacatorul nu va face interogări multiple pentru același cuvânt, iar abandonarea rulării algoritmului depinde doar de valoare lui c_i, necunoscută lui A, dar obținută cu probabilitatea $1/(q_T+1)$, reiese că probabilitate ca acesta să abandoneze execuția în timpul unei interogări pentru un cuvânt-trapă este $1/(q_T+1)$. Cum atacatorul poate face multe q_T astfel de interogări, iar acestea sunt evenimente independente reiese că probabilitatea totală ca A să abandoneze rularea este $(1-1/(q_T+1))^{q_T} \geq 1/e$

Propoziția 2: Probabilitatea ca algoritmul B să nu abandoneze în timpul provocării este de cel puțin $1/(q_T)$. Așadar $Pr[E_2] \geq 1/(q_T)$.

Demonstrație: Se știe că algoritmul va abandona în timpul provocării dacă $c_0 = c_1 = 1$. Cum Pr[$c_i = 0$] = $1/(q_T+1)$, iar evenimentele sunt independente (A nu poate să interogheze trapele celor două cuvinte W_0, W_1) $\Rightarrow$ Pr[$c_0 = c_1 = 1$] = $(1-1/(q_T+1))^2 \leq 1/(q_T)$.

Propoziția 3: Presupunând că A are la dispoziție cheia $[g,u_1]$ și că acesta semnalează că vrea să fie provocat pentru cuvinte W_0, W_1. În consecință, A îi oferă provo-

carea $C = [g^r, J]$. Astfel în timpul provocării, atacatorul face o H_2- interogare pentru $H_2(e(H_1(W_0), u_1^r))$ sau pentru $H_2(e(H_1(W_1), u_1^r))$ cu o probabilitate de cel puțin 2ε.

Demonstrație: Presupunnând că atacatorul nu face niciuna dintre aceste interogări, atunci bitul care va indica răspunsul pentru provocare va fi un eveniment independent de cunoștiințele lui A și deci probabilitatea ca $b = b'$ va fi $1/2$. Din definiția dată lui A, se știe însă că în timpul provocării $|Pr[b = b'] - 1/2| \geq \varepsilon$

$$\begin{aligned} Pr[b = b'] &= Pr[b = b'|E_3]Pr[E_3] + Pr[b = b'|\neg E_3]Pr[\neg E_3] \\ &\leq Pr[b = b'|E_3]Pr[E_3] + Pr[\neg E_3] = 1/2Pr[E_3] + Pr[\neg E_3] \\ &= 1/2 + 1/2Pr[\neg E_3] \end{aligned}$$

$$Pr[b = b'] \geq Pr[b = b'|E_3]Pr[E_3] = 1/2Pr[E_3] = 1/2 - 1/2Pr[\neg E_3]$$

$$\varepsilon \leq |Pr[b = b'] - 1/2| \leq 1/2Pr[\neg E_3] \Rightarrow Pr[\neg E_3| \geq \varepsilon$$

În eventualitatea în care B nu abandonează execuția, atacatorul A va ajunge să facă o interogare de trapă pentru $H_2(e(H_1(W_0), u_1^r))$ sau pentru $H_2(e(H_1(W_1), u_1^r))$ cu o probabilitate de cel puțin 2ε și deci, va face o intergoare pentru $H_2(e(H_1(W_b), u_1^r))$ cu o probabilitate de cel puțin ε. Ca urmare, $H_2(e(H_1(W_b), u_1^\gamma)) = e(g^{\beta+a_b}, g)^{\alpha\gamma}$ va fi prezent într-un tuplu din lista-H_2. În aceste condiții, algoritmul B va alege pereche corectă cu o probabilitate de $\varepsilon/q_{\mathrm{H}_2}$ și nu va întrerupe rularea cu o probabilitate de cel puțin $1/(q_T)$. Din toate acestea reiese că probabilitatea de succes a algoritmului B este de $\varepsilon/(eq_T q_{\mathrm{H}_2})$.

3. Tehnici de criptare care păstrează ordinea

Scopul criptării care păstrează ordinea, pe scurt OPE (engl: *Order Preserving Encryption*) este de a cripta o bază de date astfel încât valorile criptate să se afle în aceeași relație de ordine ca valorile inițiale, în senul că dacă x și y sunt două plaintexte astfel încât $x \leq y$, atunci $Enc(k,x) \leq Enc(k,y)$.

3.1. Criptare care ascunde distribuția

Pentru ca la analiza distribuției criptotextului, un atacator să nu poate deduce întreaga mulțime de valori pornind de la o anumită valoare criptată cunoscută, este necesar ca cele două seturi de date, criptate și necriptate, să aibă distribuții diferite. În aceast scop se alege o distribuție țintă după care se dorește distribuirea bazei de date criptate.

În cadrul procedurii de criptare a unei baze de date B sunt utilizate toate valorile din cadrul acesteia și o bază de date monstre de valori din distribuția țintă. Doar baza de date criptată, C este memorată pe disk. De asemenea, este necesară K- informație suplimentară utilizată pentru decriptarea informației criptate, respectiv pentru criptarea de noi valori. Această informație suplimentară este păstrată în formă criptată, folosindu-se metode de criptare standard.

În continuare va fi prezentată prima schemă de criptare care păstrează ordinea descrisă de către Agrawal și alții în [11].

Această tehnică are la bază 3 etape:

1. *Modelarea:* distribuția de intrare, respectiv cea țintă sunt modelate în funcții spline liniare discrete

2. *Aplatizarea:* textul B în clar este transformat într-o bază de date ”plată” (uniformizată) P astfel încât valorile din cadrul ei sunt uniform distribuite

3. *Transformarea:* Baza de date uniformizată P este transformată în baza de date criptată C astfel încât valorile din C sunt distribuite conform distribuției țintă.

Modelarea Pentru modelare distribuției datelor există două mari categorii de tehnici:

- bazate pe histograme - informația statistică este redată prin contorizarea datelor care se încadrează în anumite intervale

- parametrice - aproximează o distribuție fixând parametrii unei funcții

Experimentele au demonstrat că valorile din baza de date ”plată” erau într-adevăr uniform distribuite doar în cazul în care numărul ales de intervale era nerezonabil de mare. Această situație se datora în mare parte falsei presupunerii că valorile fiecărui interval sunt uniform distribuite în cadrul acestuia.

Din păcate, metodele parametrice oferă o estimare slabă pentru distribuțiile neregulate, care se presupune că vor apărea adesea în cadrul aplicației.

Cum niciuna din metode, luate în mod independent nu oferă o bună soluție pentru reprezentarea distribuției, s-a recurs la o soluție care combină cele două tehnici.

În primă fază se realizează o partiționare pe intervale a datelor, iar apoi se modelează distribuția fiecărui interval ca o funcție splină liniară. Splina unui interval $[b_l, b_h)$ nu este atceva decât linia care conectează densitățile celor două extremități ale intervalului.

De asemenea, amplitudinea valorilor poate să varieze în cadrul intervalului. Pentru a determina numărul de valori dintr-un interval se utilizează principiul lungimii minime de descriere (MDL - minimum description length).

Marginile intervalului sunt determinată în două faze:

1. *Faza de creștere.* Domeniul valorilor este împărțit recursiv în subdomenii. Fiecare partiționare a unui interval reduce deviația maximă a funcției de densitate a intervalul nou format în comparație cu intervalul părinte.

2. *Faza de reducere.* Pe baza principiului MDL, unele intervale sunt eliminate astfel încât să se minimizeze numărul de intervale, în timp ce valorile din fiecare interval rămân uniform distribuite.

Fiind dat intervalul $[b_l, b_h)$, având $h-l-1$ puncte sortate: $\{b_l+1, b_l+2, ..., b_h-1\}$ se vor efectua următorii pași:

1. Se va calcula funcția splină liniară pentru acest interval;

2. Pentru fiecare punct b_s din interval se va calcula valoarea corespunzătoare cazului în care punctele ar fi distribuite conform distribuției de densitate reprezentată de splina liniară de la punctul 1;

3. Se scindează intervalul în punctul care are cea mai mare deviație față de valoarea calculată la punctul anterior. Împărțirea intervalului se oprește în momentul în care numărul de intervale a atins o anumită valoare prestabilită (de exemplu, 10).

Principiul MDL spune că cel mai bun model pentru codificarea datelor este acela care minimizează suma dintre costul de descriere a modelului și costul de descriere a datelor în termenii modelului.

Pentru un interval dat $[p_l, p_h)$, beneficiul local al scindării intervalului în punctul p_s este dat de:

$$\begin{aligned} LB(p_l, p_h) = & DataCost(p_l, p_h) - DataCost(p_l, p_s) \\ & - DataCost(p_s, p_h) - IncrModelCost, \end{aligned}$$

unde $DataCost(p_1, p_2)$ determină costul descrierii datelor din intervalul $[p_1, p_2)$ și $IncrModelCost$ este costul suplimentat de modelare datorat partiționării unui interval în două subintervale.

Beneficiul global GB al scindării acestui interval în puctul p_s ține cont de beneficile împărțirilor recursive ce vor urma:

$$GB(p_l, p_h) = LB(p_l, p_h) + GB(p_l, p_s) + GB(p_s, p_h)$$

Dacă $GB > 0$, împărțirea se păstrează; altfel, împărțirea în punctul p_s și toate scindările recursive din intervalul $[p_l, p_h)$ sunt eliminate. Observație: Acest procedeu este aplicat în manieră *bottom-up* (de la bază spre vârf) și astfel, costul este liniar în numărul de scindări.

Se presupune momentan existența unei funcții de mapare M cu doi parametri, un coeficient pătratic și un factor de scalare, care transformă valorile extrase dintr-o funcție de densitate liniară într-o mulțime de valori uniform distribuite.

Fiind dat un interval, prin intermediul funcției M, se obține o distribuție uniformă a valorile acestuia și apoi se calculează costurile codificării devierilor de la distribuția uniformă. Fie funcția M care transformă mulțimea de valori $\{b_l + 1, b_l + 2, ..., b_h - 1\}$ în mulțimea $\{f_l, f_l + 1, ..., f_h - 1\}$.

Costul pentru codificarea unei valori $p_i \in [p_l, p_h)$ este

$$Cost(p_i) = log|f_i - E(i)|,$$

unde $E(i)$ - valoare celui de al i-lea element din cazul distribuției uniforme este dată de:

$$E(i) = f_l + \frac{i-l}{h-l}(f_h - f_l)$$

Costul codificării tuturor valorilor din intervalul $[p_l, p_h)$ este dat de

$$DataCost(p_l, p_h) = \sum_{i=l+1}^{h-1} Cost(p_i)$$

Pentru m intervale, sunt necesare $m+1$ margini, m coeficienți pătratici, m factori de scalare și în consecință, costul modelului va fi $3(m+1) \cdot 32$, unde fiecare valoare se presupune că va fi memorată pe 32 de biți. Costul unui interval adițional este: $IncrModelCost = 32 \cdot 3 = 96$

Aplatizarea În această fază se dorește ca fiecare interval de plaintext B să fie transformat într-un interval B^f astfel încât dimensiunea intervalului B^f să fie proporțională cu numărul de valori din B. Astfel, un interval dens din plaintext va fi extins, iar un interval întins va fi compresat. Valorile dintr-un interval sunt mapate în așa fel încât densitatea din intervalul aplatizat să fie uniformă. Cum densitatea din fiecare interval și cea dintre intervale sunt uniforme, valorile din baza de date aplatizată vor fi uniform distribuite.

Dacă o distribuție peste $[0, p_h)$ are funcția de densitatea $qp + r$, unde $p \in [0, p_h)$, atunci pentru orice constantă $z > 0$ funcția de mapare:

$$M(p) = z(\frac{q}{2r}p^2 + p)$$

va conduce la o mulțime de valori uniform disribuite.

Aceasta datorită faptului că panta funcției de mapare în orice punct p este proporțională cu densitatea în punctul p:

$$\begin{aligned} \frac{dM}{dp} &= \frac{z}{r}(qp + r) \\ &\propto qp + r \end{aligned}$$

Fie coeficientul pătratic $s = q/2r$. Astfel

$$M(p) = z(sp^2 + p).$$

Factorul de scalare z diferă de la un interval la altul pentru ca în final, densitatea dintre intervale să fie uniformă.

Așadar factorul de scalare z pentru fiecare interval B este necesar pentru ca:

1. Oricăror două valori diferite din plaintext să le corespundă două valori diferite din spațiul de valori uniformizat și astfel, actualizarea valorilor să se poată realiza incremental.

   $$\forall p \in [0, w) : M(p+1) - M(p) \geq 2.$$

   Motivul pentru care spațiul dintre valori este 2 și nu 1 este de a face criptarea tolerantă la erorile de aproximare a numerelor de tip float. Dacă M este scris explicit, se obține:

   $$\forall p \in [0, w) : z \geq 2/(s(2p+1)+1).$$

Se observă că valoare maximă a lui z diferă în funcție de semnul lui s și astfel:

$$\hat{z} = \begin{cases} 2, & \text{dacă } s \geq 0 \\ 2/(s(2w-1)+1), & \text{dacă } s < 0 \end{cases}$$

2. Fiecărui interval îi corespunde un spațiu proporțional cu numărul de puncte n din interval. De exemplu, dacă w este lățimea intervalului și $w^f = M(w)$ este lățimea intervalului după uniformizare, atunci $w^f \propto n$. Pentru a satisface această condiție ete necesar ca petru fiecare interval:

$$w^f = Kn$$

Fie lățimea minimă pentru fiecare interval:

$$\hat{w}^f = \hat{z}(sw^2 + w)$$

$$K = max[\hat{w}_i^f], i = 1, ..., m$$

Factorul de scalare

$$z = \frac{Kn}{sw^2 + w}$$

va satisface ambele condițiile.

Cheie de criptare K^f este calculată o singură dată la criptarea bazei de date. În timp ce noi valori sunt adăugate în baza de date, K^f este utilizată, dar nu și recalculată.

Fie baza de plaintexte P, respectiv baza de date uniformizată F date prin $[p_{min}, p_{max})$ și $[f_{min}, f_{max})$.

Obervație:

$$f_{max} = f_{min} + \sum_{i=1}^{m} w_i^f$$

unde $w_i^f = M_i(w_i)$ întrucât w_i este lungimea domeniului în clar B_i și w_i^f lungimea intervalului uniformizat corespunzător.

Pentru a stabili valoarea dintr-un interval uniformizat corespunzătoare unei valori date p, se va determina intervalul B_i căruia îi aparține, utilizându-se informațiile despre marginile intervalului salvate în K^f. Un punct p este asociat valorii f, din domeniul uniformizat folosind ecuația

$$f = f_{min} + \sum_{j=1}^{i-1} w_j^f + M_i(p - p_{min} + \sum_{j=1}^{i-1} w_j)$$

Se poate rescrie ecuația:

$$p = p_{min} + \sum_{j=1}^{i-1} w_j + M_i^{-1}(f - f_{min} + \sum_{j=1}^{i-1} w_j^f),$$

unde

$$M_i^{-1}(f) = \frac{-z \pm \sqrt{z^2 \pm 4szf}}{2zs},$$

iar z si s reprezintă factorul de scalare si respectiv coeficientul pătratic al funcției de mapare M.

Așadar este necesară informația din K^f pentru a determina intervalul B_i^f căruia îi aparține valoare f și apoi se aplică funcția M^{-1}. Din cele două valori posibile pentru M^{-1}, doar una se va afla între granițele intervalului.

Se observă că atât $M(p)$ cât și M^{-1} nu sunt de obicei valori întregi și de aceea vor fi rotunjite la cel mai apropiat întreg.

Pentru a reduce șansele de apariție a erorilor de calcul cauzate de aproximarea valorilor raționale se va verifica dacă $M^{-1}(f) = p$ imediat după calculul lui $M(p)$.

Dacă reiese că $M^{-1}(f)$ este de fapt aproximat la $p-1$, se va cripta p ca $f+1$, în loc de f. Din moment ce se asigură faptul că cele două valori consecutive sunt cu cel

puțin la distanță 2 de spațiul uniformizat atunci când se calculează factorul de scalare, $M^{-1}(f)+1$ va fi decriptat ca p și nu ca $p+1$. În mod similar, dacă $M^{-1}(f)=p+1$, p va fi decriptat ca $f+1$.

Transformarea Pasul de transformare este foarte asemănător cu pasul de uniformizare. Avem o mulțime uniform distribuită de valori, iar scopul acestei etape este de a afla valorile corespunzătoare din distribuția țintă. Pentru aceasta se va calcula o altă mulțime de valori distribuite uniform, de data aceasta pornind de la distribuția țintă, dorindu-se ca această mulțime să se "suprapună" peste cea obținută din valorile bazei de date inițiale.

Așadar acest proces presupune:

1. Calculul intervalelor folosite pentru uniformizarea distribuției.
2. Pornind de la distribuția țintă, calculul intervalele pentru distribuția uniformă $\hat{B}^f$.
3. Scalarea intervalelor distribuției țintă uniformizate, calculate la pasul 1, și scalează distribuția standard în mod proporțional.

Intervalele corespunzătoare distribuției plaintextului există deja (pasul 1) și se va împărți distribuția țintă în intervale, în mod independent de distribuția plaintextului(pasul 2).

Se scalează distribuția țintă (și distribuția țintă "aplatizată") în așa fel încât amploarea distribuției generate uniformizându-se distribuția țintă scalată să devină egală cu amploarea distribuției generate de uniformizarea distribuției textului în clar (pasul 3).

Modelarea distribuției țintă duce la obținerea unei mulțimi de intervale $\{B_1^t,...,B_k^t\}$.

Pentru fiecare interval B^t de lungime w^t vom avea de asemenea o funcție de mapare M^t și parametrii asociați s^t și z^t.

Pentru a calcula factorul de scalare z^t al fiecărui interval, se va folosi o procedură similară cu cea prezentată, cu excepția faptului că prima condiție este inversată. În acest punct este necesar să se asigure faptul că două valorile consecutive din spațiul "plat" vor fi asociate cu două valori distincte din spațiul țintă (așa cum mai devreme a fost acoperită situația în care două valori consecutive din spațiul în clar au fost asociate cu două valori distincte din spațiul "aplatizat").

O analiză similare celei de dinainte condue la:

$$z^t = \frac{K^t n^t}{s^t (w^t)^2 + w^t}$$

unde

$$K^t = min\left[\frac{\hat{z}_i^t (s^t (w^t)^2 + w^t)}{n_i^t}\right], i = 1, ..., k$$

și

$$\hat{z}^t = \begin{cases} 0.5/(1 + s^t(2w^t - 1)), & \text{dacă } s^t > 0 \\ 0.5, & \text{dacă } s^t \leq 0 \end{cases}$$

Fie $\hat{B}^f$ intervalul din spațiul uniformizat corespunzător intervalului B^t, de lungime $\hat{w}^f$ și $\{B_1^f, ..., B_m^f\}$ intervalele din distribuția "plată" a plaintextului.

Fie intervalul B^f de lungime w^f. Se dorește ca dimensiunea celor două distribuții uniforme să fie egală și de aceea se va defini un factor de asociere L ca fiind:

$$L = (\sum_{i=1}^{m} w_i^f)/(\sum_{i=1}^{k} \hat{w}_i^f)$$

Apoi, ambele intervale, B^t și intervalul țintă uniformizat $\hat{B}^f$, sunt scalate cu factorul L. Așadar dimensiunea intervalul cripat B^c, corespunzător intervalului țintă B^t, este dat de $w_i^c = Lw_i^t$, iar dimensiunea intervalului uniformizat scalat $\bar{B}^f$ este dată de $\bar{w}^f = L\hat{w}^f$.

Pentru a mapa valorile din intervalul B^c cu cele din intervalul $\bar{B}^f$ este definită funcția M^c.

Coeficientul pătratic al acesteia este definit ca $s^c = s^t = L$ și factorul de scalare z^c este setat ca z^t.

Se știe că $s = q^t/2r^t$, unde $n^t = q^t x + r^t$ este aproximativ densitatea intervalului B^t. Atunci când se extinde domeniul cu un factor L, $q^t = r^t$ este redus cu factorul L. De aceea, $s^c = s^t = L$.

În acest moment z^c ar trebuie să asigure că $M^c(w^c) = \bar{w}^c$, ceea ce se obține fixând $z^c = z^t$ întrucât:

$$\begin{aligned} M^c(w^c) &= z^c(s^c(w^c)^2 + w^c) \\ &= z^t((s^t/L)(Lw^t)^2 + Lw^t) \\ &= LM^t(w^t) \\ &= L\hat{w}^f \\ &= \bar{w}^f \end{aligned}$$

Marginile intervalului spațiului criptării au fost salvate în structura K^c. Pentru fiecare interval, sunt salvate coeficientul pătratic s^c și factorul de scalare z^c.

O valoare din intervalul B^f poate fi asociată cu o valoare criptată c folosind ecuația:

$$c = c_{min} + \sum_{j=1}^{i-1} w_j^c + (M_i^c)^{-1}(f - f_{min} + \sum_{j=1}^{i-1} \bar{w}_j^f),$$

unde

$$(M^c)^{-1}(f) = \frac{-z \pm \sqrt{z^2 + 4szf}}{2zs}$$

Doar una dintre cele două valori posibile va aparține intervalului asociat criptotextului, iar valoarea întoarsă de $(M^c)^{-1}$ va fi rotunjită.

O valoare criptată c din intervalul B_i^c va corespunde unei valori din distribuția uniformă, f datorită ecuației:

$$f = f_{min} + \sum_{j=1}^{i-1} \bar{w}_j^f + M_i^c(c - c_{min} - \sum_{j=1}^{i-1} w_j^c)$$

Dimensiunea criptotextului depinde de dimensiunea plaintextului și de distribuția țintă.

Fie g_{min}^p cea mai mică diferență dintre valorile consecutive sortate din datele de intrare și respectiv, g_{max}^p cea mai mare diferență. În mod similar, fie g_{min}^t și g_{max}^t cea mai

mică și respectiv, cea mai mare diferență dintre datele consecutive sortate din distribuția țintă.

Se definește $G^p = g^p_{max}/g^p_{min}$ și $G^t = g^t_{max}/g^t_{min}$. Numărul suplimentar de biți necesari pentru criptare în cel mai nefavorabil caz poate fi aproximat ca $G_p + log(G_t)$, ceea ce este echivalent cu o margine superioară pentru $c_{min} - c_{max}$ dată de $G_p \times G_t \times (p_{max} - p_{min})$.

Se observă faptul că atunci când se realizează uniformizarea este necesar ca toate spațiile dintre valori să devină egale. Dacă majoritatea diferențelor din plaintext sunt aproape de g^p_{min} în timp ce doar câteva sunt aproape de g^p_{max}, va fi necesar să se mărească fiecare diferență până la g^p_{max} astfel încât g^p_{max}/g^p_{min}.

Analog, poate exista o mărire a t^p_{max}/t^p_{min}, atunci când are loc transformarea, dacă majoritatea diferențelor dintre valori se apropie de t^p_{max}

Observație: Se poate controla în mod explicit G_t, din moment ce distribuția țintă este una aleasă. Deoarece G_p nu poate fi controlată direct este de așteptat ca $G_p \times G_t$ să fie mult sub pragul de 2^{32} ceea ce înseamnă că vor fi necesari 4 biți suplimentari pentru datele de ieșire (criptate) față de datele de intrare.

3.2. Criptare mutabilă care păstrează ordinea

În acestă secțiune va fi prezentată schema de criptare mutabilă care păstrează ordinea (sau *mOPE*, engl: *mutable Order Preserving Encryption*) propusă de Ada Popa și colaboratorii săi în [12].

Ideea de bază este ca criptotextele să fie stocate la server într-o structură de tip *arbore binar de căutare*. Un arbore binar de căutare este un arbore în care, pentru fiecare nod de valoare v, valorile nodurilor din subarborele stâng sunt strict mai mici decât v, iar cele din subarborele drept sunt strict mai mari decât v.

Schema folosește un arbore $B-tree$ din motive de eficiență: timpul de inserare, ștergere sau căutare, în cel mai rău caz, este logaritmic și facilitează actualizarea eficientă a criptotextelor.

Fie $DET = (DET.KeyGen, DET.Enc, DET.Dec)$ o schemă de criptare formată din algoritmi determiniști. În practică, DET poate fi implemetată prin orice schemă standard de criptare bloc precum AES (engl: *Advanced Encryption Standard*), având însă un vector de inițializare constant.

Fiecare nod din arbore conține criptarea DET a unei valori cu cheia secretă sk, dar locația fiecăruia depinde de valoarea plaintextelor. Serverul nu cunoaște decât criptotextele și de aceea, aranjarea arborelui va fi realizată cu ajutorul clientului. Acesta va ghida serverul spre locația unde va fi inserată fiecare nouă valoare adăugată.

Inițial, serverul va returna clientului, criptotextul corespunzător rădăcinii, pe care clientul îl va decripta, va compara valoarea curentă cu aceea din rădăcină și va cere serverului să trimită în continuare valoare rădăcinii subarborelui stâng, dacă numărul este mai mic decât valoarea rădăcinii, sau valoarea din subarborele drept dacă numărul curent este mai mare decât cel din rădăcină. Procesul se repetă până când se ajunge în locul în care valoarea curentă poate fi inserată, iar atunci aceasta este criptată și trimisă la server. Astfel, clientul va dezvălui serverului doar informația referitoare la relația de ordine.

Algoritmul de inserare a unei valori în arbore este următorul:

Inserare (v)

1. $Cl \leftrightarrow Ser$: Clientul cere serverului criptotextul c' aflat în rădăcina arborelui

2. Cl: Clientul decriptează c' și obține v'

3. $Cl \leftrightarrow Ser$: Dacă $v < v'$ clientul transmite serverului "stânga"; Dacă $v > v'$ clientul transmite serverului "dreapta";

4. Ser: serverul returnează următoarul criptotext c'', în funcție de informația primită de la client și se întoarce la pasul 2.

5. Algoritmul se oprește atunci când v este găsit sau când serverul ajunge la un nod gol în arbore. Rezultatul algorimului este format din pointerul din arborele OPE, drumul din arborele OPE de la rădăcină și dacă v a fost găsit sau nu.

Se observă că drumul de la rădăcina la o valoare din arbore indică ordinea relativă a unui nod în comparație cu celelalte noduri. Dacă fiecare muchie stângă ar fi etichetată cu "0", iar fiecare muchie dreaptă etichetată cu "1", prin concatenarea etichetelor muchiilor de la rădăcină până la o valoare dintr-un nod s-ar obține reprezentarea binară a unui număr. Numerele obținute din reprezentările binare corespunzătoare traseului rădăcină-nod pentru fiecare nod se vor afla în aceeași relație de ordine ca plaintextele asociate fiecărui nod.

Problema care apare este că nodul rădăcină este un șir gol și deci, nu poate fi pus într-o relație de ordine cu alte numere. Pentru a rezolva acest impediment se vor aduce toate numerele la aceeași dimensiune. Așadar, fie $path$ drumul de la rădăcină până la o valoare dată din arborele binar, atunci reprezentarea binară obținută va fi:

$$codificareOPE = [path]10\ldots0.$$

Numărul va fi completat cu "0"-uri până la o dimensiune m convenabilă.

Pentru a evita cazul în care codificările OPE ar deveni prea mari, schema trebuie să mențină o înălțime logaritmică a arborelui, ceea ce implică operații ocazionale de rebalansare. De exmplu, într-un B-plus arbore, dacă un nod conține prea multe elemente, acesta va trebui împărțit în două, iar fiecare nou nod va avea noduri copil noi. Dacă nodul părinte conține de asemenea, prea multe valori, împărțirea se propagă în sus.

Aceste balansări ale arborelui definesc "mutabilitatea" codificării OPE, deoarece după o rebalansare, nodul poate fi mutat într-o altă parte a arborelui, altfel modificându-și *path*-

ul. Posibilitatea de a muta codificările OPE asigură menținerea în cadre rezonabile a dimensiunilor acestora.

După balansarea arborelui OPE, serverul trebuie să reactualizeze orice valoare pe care o stochează sub formă de codificare OPE.

Clientul necesită resurse de ordinul $O(logn)$, unde n este numărul total de valori codificate, iar codificările necesită și ele $O(logn)$ deoarece arborele are înălțime logaritmică.

Clientul nu este implicat în rebalansare: deși serverul nu cunoaște valorile din spatele codificărilor, el are nevoie doar de informația referitoare la relația de ordine dintre acestea.

Costul serverului pentru a actualiza valorile.

Costul logaritmic al arborilor de căutare binar este calculat deobicei luând în considerare doar numărul de noduri implicate în rebalansare. Totuși numărul de criptotexte afectate este numărul de copii din subarborii mutați în timpul rebalansării și care poate fi asimptotic mai mare. De exmplu, dacă un nod aflat în partea superioară a arborelui se mută, doar un număr mic de noduri vor fi afectate de rebalansare, dar toți copii acestor noduri își vor schimba codificarea.

Există riscul ca un client să obțină o codificare de la server, iar mai apoi acesta să realizeze operații de rebalansare și astfel, codificarea pe care acesta o deține să nu mai corespundă poziției din arbore.

Pentru a preveni o astfel de situație, se va introduce o nouă componentă, un tabel OPE sau $OPETable$. Atunci când un nou element este introdus în arbore, serverul rebalansează arborele, tabelul va fi actualizat astfel încât să asocieze criptotextele cu noile codificări sau codificările actualizate. Folosind acest tabel, clientul nu va avea nevoie să țină evidența criptotextelor deoarece acestea rămân mereu valabile.

Având un criptotext, serverul va putea să calculeze codificare OPE oricând, fără ajutorul clientului. Tabelul OPE este util și în situația în care un client criptează aceeași valoare de mai multe ori, economisind un nou efort de calcul.

Schema funcționează după cum urmează:

- *KeyGen* [rulează la Client]
  intrare : parametrul de securitate s
  ieșire : $sk \leftarrow DET.KeyGen(1^s)$

- *InitState* [rulează la Server]
  intrare: parametrul de securitate s
  Inițializează starea st a serverului cu arborele OPE și tabelul OPE conținând doar valorile $\pm\infty$
  ieșire : stare st

- *Enc* [rulează la Client și la Server]
  intrare: cheia sk, valoare v, starea st

  1. Client: $c \leftarrow DET.Enc(sk, v)$ și trimite c la Server
  2. Server: dacă c se află în tabelul OPE, Server: returnează st neschimbată; Client: returnează c
  3. Server: Altfel,
     (a) $Client \leftrightarrow Server$ parcurge transversal Arborele OPE (Alg 1.) astfel încât să se obțină $path$-ul lui c. Serverul calculează codificarea OPE lui c ca în (1) și o stochează în tabelul OPE
     (b) Server: Dacă arborele OPE are nevoie de rebalansare și actualizare după inserare, se rebalansează arborele și se actualizează
     (c) Server: returnează noua stare Client: returnează c

- *Dec* [rulează la Client]
  intrare: cheia sk, codificare c *ieșire*: $v \leftarrow DET.Dec(sk, c)$

- *Order*
  intrare: starea st, codificare c
  ieșire: Dacă c se află în tabelul OPE returnează codificarea OPE corespunzătoare din tabel, altfel semnalează eroare

Din moment ce schema de criptare poate muta criptotexte, va avea o sintaxă diferită față de o schemă obișnuită de criptare, în sensul în care, criptotextul corespunzător valorii v poate fi de două tipuri:

- criptotext permanent *c*: corespunde lui *v* și nu se schimbă (obținut prin criptare de tip DET)
- o codificare tranzitorie care păstrează relația de ordine: o codificare corepsunzătoare unei anumite valori se schimb de-a lungul timpului, atunci când arborele este rebalansat. Este denumită codifare OPE.

Definiția 1: O *schemă de criptare mutabilă care păstrează ordinea* sau mOPE (engl: *mutable Order Preserving Encription*) pentru domeniul de plaintexte *D* este un tuplu de algoritmi polinomiali *mOPE = (KeyGen, InitState, Enc, Dec, Order)* rulat de client și un server care trece prin stări, unde *KEyGen* este un algoritm probabilist, iar restul sunt deterministe, în timp ce *Enc* este interactiv.

- *KeyGen* (rulează la Client) *intrare*: parametrul de securitate *s ieșire*: cheia secretă *sk*
- *InitState* (rulează la server) *intrare*: parametrul de securitate *s ieșire*: starea inițială *st*
- *Enc* (algoritm interactiv între Client și Server) *intrare Client*: cheia secretă *sk* și valoarea *v intrare Server*: starea *st intrare Client*: criptotextul *c intrare Server*: starea st'
- *Dec*: (rulează la Client) *intrare*: cheia secretă *sk* și criptotextul *c ieșire*: valoarea *v*
- *Order* (rulează la Server) *intrare*: stare *st* și criptotextul *c ieșire*: codificare OPE *e*

Timpul de rulare a *Enc* este polinomial în $|sk|$ și $|v|$.

În această schemă $Order(st, c)$ produce codificare OPE a valorii corespunzătoare criptotextului *c*. În practică $Order(st, c)$ se va schimba rar prin starea *st* deoarece un arbore se rebalanează rar. De aceea, în practică codificările OPE pot fi stocate pe disk și actualizate atunci când are loc o rebalansare.

Pentru a demonstra corectitudinea schemei, este nevoie ca algoritmul de decriptare să decripteze corect valorile, iar algoritmul *Order* să ofere criptotexte care să mențină relația de ordine între elemente.

Se consideră secvența $seq = v_1, v_2, \ldots, v_n$. Starea se modifică după fiecare criptare de la starea st_0 la starea st_n, calculând în mod succesiv $(c_i, st_i) \leftarrow Enc(sk, v_i, st_{i1})$, pentru $i = 1 \ldots n$, unde $st_0 \leftarrow InitState(1^s)$

Definiția 2: *O schemă mOPE pentru domeniul de plaintexte D este corectă dacă, pentru orice parametru de securitate s și pentru orice cheie* $sk \leftarrow KeyGen(1^s)$

1. *Pentru toți* $v \in D$ *și pentru toate stările st, pentru orice c returnat de* $Enc(sk, v, st)$, $DEc(sk, c) = v$

2. *Pentru orice secvență* $seq = \{v_1, \ldots, v_n\} \in D^n$, *pentru toate perechile* $v_i, v_j \in seq$, *pentru orice* c_i, c_j *obținute ca mai sus, avem*

$$v_i < v_j \equiv Order(st_n, c_i) < Order(st_n, c_j)$$

Într-o schemă mOPE ideală nu ar trebui să se dezvăluie mai mult decât relația de ordine dintre criptotexte, adică un atacator nu poate face diferența între două criptotexte a două secvențe de valori atâta timp cât au aceeași relație de ordine.

Jocul de securitate dintre un client *Cl* și un *Adv* pentru un parametru de securitate *s* are loc astfel:

1. Clientul *Cl* generează $sk \leftarrow KeyGen(1^s)$ și alege un bit aleator *b*;

2. Clientul *Cl* și adversarul *Adv* interacționează într-un număr polinomial de runde, în cadrul cărora adversarul este adaptativ. La runda *i*:

   a Adversarul *Adv* trimite valorile $v_i, v_j \in D$, la clientul *Cl*;

   b Clientul *Cl* interacționează cu serverul *Ser* în cadrul rulării algoritmului *Enc* având la intrare cheia *sk* și valoarea v_i, în timp ce *Adv* monitorizează toate stările serverului

3. Adversarul *Adv* returnează b'

Adversarul *ADv* câștigă jocul dacă ghicește corect ($b = b'$) și dacă secvențele $\{{v_i}^0{}_i$ și $\{{v_i}^1{}_i$ au aceeași relație de ordine, adică pentru toți $i, j, {v_i}^0 < {v_j}^0 \equiv {v_i}^1 < {v_j}^1$.

Fie $win^{Adv,s}$ o variabilă aleatoare care semnifică succesul adversarului în jocul prezentat.

Definiția 3: *O schemă mOPE este IND-OCPA sigură dacă pentru toți adversarii probabiliști polinomiali Adv și pentru toți parametrii de securitate suficient de mari s,*

$$Pr[win^{Adv,s} \leq \frac{1}{2} + negl(s)]$$

Teorema 1: *Schema mOPE prezentată este IND-OCPA sigură.*

Demonstrație: Se consideră un adversar Adv și orice două secvențe de valori pentru care Avd le cere în timpul jocului de securitate: $v = (v_1, \ldots, v_n)$ și $w = (w_1, \ldots, w_n)$. Ceea ce Adv observă este informația pe care serverul o primește în timpul jocului de securitate.

Primul pas este de a se utiliza securitatea garantată de schema de criptare DET și se presupune că criptotextele obținute prin DET sunt indistinctibile din punct de vedere computațional de valori aleatoare care au același tipar de repetiție (de exmplu, produse de un oracol aleator).

La următorul pas, se examinează informația pe care adversarul o obține în cazul în care clientul criptează valoarea v și în cazul în care criptează w și se poate arăta că această informație este teoretic aceeași.

Pentru aceasta, se procedează inductiv după numărul de valori care urmeaz a fi criptate.

Cazul de bază este atunci când nici o valoare nu este criptată, iar adversarul începe cu aceeași informație.

Se consideră că după i criptări, Adv obține aceeași informație în ambele cazuri și se arată că după pasul $i+1$, informația rămâne aceeași. La pasul $i+1$, Cl și Adv rulează algoritmul Enc pentru v_i și w_i.

Există două posibilități.

- Codificărea lui v_i se află în tabelul OPE. Mai apoi codificare lui w_i se află de asemenea în tabelul OPE (și vice versa) deoarece vși w se află în aceeași relație de ordine.

  Mai exact, $v_i = v_j$ dacă $w_i = w_j$ și deci tiparul repetițiilor va fi același. În acest caz, Cl nu oferă nicio informație adversarului Adv.

- codificările lui v_i (și de aceea, a lui w_i) nu se sflă în tabelul OPE. *Cl* și *Adv* interacționează cu algoritmul *Enc* în ambele cazuri. Cum v și w se află în aceeași relație de ordine, parcurgerea arborelui de la rădăcină la nod trebuie să fie aceeași și în cazul clientului *Cl*, cât și în cel al adversarului *Adv*.

  De asemenea, informația pe care clientul o oferă serverului este legată de ce muchie să aleagă în timpul parcurgerii, informație care este aceeași în ambele cazuri.

Astfel, *Adv* obține aceeași informație în ambele cazuri și de aceea, nu poate face distincția.

3.3. Criptare care dezvăluie ordinea

În continuare vom prezenta o schemă de criptare care dezvăluie relația de ordine (sau ORE, engl: *Order Revealing Encryption*), propusă în [13] pentru o mulțime de numere întregi pozitive, pe n biți cu următoarea funcție de scurgere de informație:

$$Fie\ \mathcal{L}(m_1,\ldots,m_t) = \{(ind_{dif}(m_i,m_j), 1\cdot(m_i < m_j)) : 1 \le i < j \le t\},$$

unde $ind_{dif}(x,y)$ oferă indexul primului bit pentru care $x \neq y$. Dacă $x = y$ atunci $ind_{dif}(x,y) = n+1$. Cu alte cuvinte, pentru $x \neq y$, dacă $x = x_1 \ldots x_n$ și $y = y_1 \ldots y_n$, atunci $ind_{dif}(x,y)$ are valoare celui mai mic index l pentru care $x_l \neq y_l$.

Pentru construcție se alege un parametru de securitate $s \in \mathbb{N}$ și un număr întreg $M \geq 3$. Fie o funcție $f : \{0,1\}^r \times ([n] \times \{0,1\}^n) \to \mathbb{Z}_M$ o funcție pseudo-aleatoare (PRF) sigură (a se vedea secțiunea *Primitive Criptografice*).

Schema *ORE = (KeyGen, Enc, Comparare)* se definește după cum urmează:

- *KeyGen*:

  intrare: parametrul de securitate s

  ieșire: cheia secretă $sk = k$, unde k este cheia pentru funcția pseudo-aleatoare f și este generată în mod uniform aleator

- *Enc*

  intrare: cheia sk și mesajul m
  Fie $b_1 \ldots b_n$ reprezentarea binară a lui m, iar $sk = k$. Pentru fiecare $i \in [n]$, se va calcula

  $$u_i = f(k, (i, b_1 b_2 \ldots b_{i-1} || 0^{n-i})) + b_i \pmod M$$

  ieșire: tuplul $(u_1, u_2, \ldots, u_n)$

- *Comparare*

intrare: criptotextele c_1 și c_2

Se va calcula

$$c_1 = (u_1, u_2, \ldots, u_n)$$
$$c_2 = (u'_1, u'_2, \ldots, u'_n),$$

unde $u_1, u_2, \ldots, u_n, u'_1, u'_2, \ldots, u'_n \in \mathbb{Z}_M$. Fie i cel mai mic index pentru care $u_i \neq u'_i$.

ieșire: $\begin{cases} 0, & \text{dacă nu există un astfel de index sau există, dar } u'_i \neq u_i + 1 \ (mod\ M) \\ 1, & \text{dacă } u'_i = u_i + 1 \ (mod\ M) \end{cases}$

Fixând un parametru de securitate s, schema prezentată, definită peste un domeniu ordonat $\mathcal{D}$ este corectă dacă $sk \leftarrow KeyGen(1^s)$ și orice mesaje $m_1, m_2 \in \mathcal{D}$,

$$Pr[Comparare(c_1, c_2) = 1 \cdot (m_1 < m_2)] = 1 - negl(s),$$

unde $c_1 \leftarrow Enc(sk, m_1)$ și $c_2 \leftarrow Enc(sk, m_2)$.

Teorema 1: *Schema ORE prezentată este corectă.*

Dmonstrație:

Se aleg două mesaje $m_1, m_2 \in \{0,1\}^n$. Pentru corectitudine este necesar ca pentru orice $sk \leftarrow KeyGen(1^s)$ și $c_i \leftarrow Enc(sk, m_i)$, cu $i \in \{1,2\}$,

$$Pr[Comparare(c_1, c_2) = 1 \dot{(} m_1 < m_2)] = 1;$$

Fie $k = sk$ o valoare pseudo-aleatoare, returnată de algoritmul de generare de chei *KeyGen*, $b_1 \ldots b_n$ reprezentarea binară a lui m_1 și $b'_1 \ldots b'_n$ reprezentarea binară a lui m_2. Se obțin $c_1 = (u_1, u_2, \ldots, u_n)$ și $c_2 = (u'_1, u'_2, \ldots, u'_n)$. Se va demonstra că $m_1 < m_2 \Leftrightarrow Comparare(c_1, c_2) = 1$.

- "$\Rightarrow$"

  Presupunând că $m_1 < m_2$, atunci există $i \in [n]$ astfel încât $b_i = 0$ și $b'_i = 1$ și pentru orice $j < i, b_j = b'_j$. Din construcția lui c_1 și c_2, pentru toți $j < i$,

$$\begin{aligned} u_j &= f(k, (j, b_1 b_2 \ldots b_{j-1} || 0^{n-j})) + b_j \\ &= f(k, (j, b'_1 b'_2 \ldots b'_{j-1} || 0^{n-j})) + b'_j = u'_j \pmod M \end{aligned}$$

Cum $b_i = 0$ și $b'_i = 1$, atunci

$$u'_i = f(k,(j,b'_1b'_2\ldots b'_{i-1}||0^{n-i})) + b'_i = u_i + 1 \pmod M,$$

iar în acest caz $Comparare(c_1,c_2)$ returnează 1.

- "$\Leftarrow$"
  Presupunând că Comparare(c_1,c_2) returnează 1. Asta înseamnă că există un index $i \in [n]$ astfel încât $u'_i = u_i + 1 \pmod M$ și pentru orice $j < i, u_j = u'_j$. Se va demonstra prin inducție după j că pentru orice $1 \leq j < i, b_j = b'_j$. Pentru $j = 1$,

  $$F(k,(1,0^{n-1})) + b_1 = u_1 = u'_1 = F(k,(1,0^{n-1}) + b'_1 \pmod M,$$

  de unde se deduce că $b_1 = b'_1$. Dacă $b_1b_2\ldots b_j = b'_1b'_2\ldots b'_j$, atunci

  $$u_{j+1} = f(k,(j+1,b_1b_2\ldots b_j||0^{n-j-1})) + b_{j+1} \pmod M$$
  $$u'_{j+1} = f(k,(j+1,b'_1b'_2\ldots b'_j||0^{n-j-1})) + b'_{j+1} \pmod M$$

  Din ipoteza inductivă $b_1b_2\ldots b_j = b'_1b'_2\ldots b'_j$ și din faptul că $u_{j+1} = u'_{j+1}$, se obține că $b_{j+1} = b'_{j+1}$. Astfel, din inducția după j, rezultă pentru toți $j < i, b_i = b'_i$.

  Pentru a completa demonstrația mai este necesar să se arate că $b_i = 0$ și respectiv, $b'_i = 1$. Presupunând că $u'_i = u_i + 1 \pmod M$ și cum se știe că $b_1b_2\ldots b_{i-1} = b'_1b'_2\ldots b'_{i-1}$, rezultă că $b'_i = b_i + 1 \pmod M$. Pentru că $M \geq 3$ și $b_i, b'_i \in \{0,1\}$, singura soluție este pentru $b_i = 0$ și $b'_i = 1$.

Noțiunea de securitate considerată pentru analiza securității acestei scheme este parametrizată de funcția de scurgere de informație $\mathcal{L}$ și se bazează pe un scenariu de simulare după cum este prezentat în cele ce urmează.

Definiția 1: *Fiind dat parametrul de securitate $s \in \mathbb{N}$, fie schema $ORE = (KeyGen, Enc, Comparare)$ o schemă de criptare care dezvăluie relația de ordine. Se consideră $A = (A_1,\ldots,A_q)$, un adversar cu $q \in \mathbb{N}$, $S = (S_0,S_1,\ldots,S_q)$ un simulator și $\mathcal{L}(\cdot)$ o funcție de scurgere de informație. Se definesc experimentele $REAL_A^{ORE}(s)$ și $SIM_{A,S,\mathcal{L}}^{ORE}(s)$ astfel:*

$REAL_A^{ORE}(s)$:

1. $sk \leftarrow KeyGen(1^s)$
2. $(m_1, st_A) \leftarrow A_1(1^s)$
3. $c_1 \leftarrow Enc(sk, m_i)$
4. pentru $2 \leq i \leq q$:
   (a) $(m_i, st_A) \leftarrow A_i(st_A, c_1, \ldots, c_{i-1})$
   (b) $c_i \leftarrow Enc(sk, m_i)$
5. returnează $(c_1, \ldots, c_q)$ și st_A

$SIM_{A,S,\mathcal{L}}^{ORE}(s)$:

1. $st_S \leftarrow KeyGen(1^s)$
2. $(m_1, st_A) \leftarrow A_1(1^s)$
3. $(c_1, st_S \leftarrow S_1(st_S, \mathcal{L}(m_1))$
4. pentru $2 \leq i \leq q$:
   (a) $(m_i, st_A) \leftarrow A_i(st_A, c_1, \ldots, c_{i-1})$
   (b) $(c_i, st_S) \leftarrow S_i(st_S, \mathcal{L}(m_1, \ldots, m_i))$
5. returnează $(c_1, \ldots, c_q)$ și st_A

Se spune că schema *ORE*, care are funcția de scurgere de informație $\mathcal{L}(\cdot)$ este o schemă sigură dacă pentru orice adversar polinomial $A = (A_1, \ldots, A_q)$, unde $q = poly(s)$, există un simulator de mărime polinomială $S = (S_0, S_1, \ldots, S_q)$, astfel încât rezultatele celor două distribuții $REAL_A^{ORE}(s)$ și $SIM_{A,S,\mathcal{L}}^{ORE}(s)$ să fie computațional indistinctibile.

Observație: Dacă funcția $\mathcal{L}$ definită ca:

$$\mathcal{L}(m_1, \ldots, m_t) = \{1 \cdot (m_i < m_j) : 1 \leq i < j \leq t\},$$

iar schema *ORE* care are această funcție de scurgere de informație este sigură, atunci ea este *IND-OCPA* sigură.

Teorema 2: *Dacă schema prezentată are funcția de scurgere de informație $\mathcal{L}_f$, definită ca mai sus și funcția pseudo-aleatoare f este sigură, atunci schema ORE prezentată este sigură.*

Demonstrație:

Fie s parametru de securitate și $A = (A_1, \ldots, A_q)$, unde $q = poly(s)$, un adversar eficient pentru jocul de securitate prezentat în Definiția 1. Se consideră un simulator eficient $S = (S_0, \ldots, S_q)$, pentru care distribuțiile de ieșire ale $REAL_A^{ORE}(s)$ și $SIM_{A,S,\mathcal{L}_f}^{ORE}(s)$ sunt indistinctibile din punct de vedere computațional.

Se definesc experimentele:

- H_0: acesta este experimentul real $REAL_A^{ORE}(s)$

- H_1: la fel ca H_0, cu excepția faptului că în timpul rulării *KeyGen*, este aleasă o funcție $F \xleftarrow{R} Fun[([n] \times \{0,1\}^{n-1}), \mathbb{Z}_M]$. În toate invocările *KeyGen*, funcția $f(k,\cdot)$ va fi înlocuită cu $F(\cdot)$.

Experimentele H_0 și H_1 sunt indistinctibile din punct de vedere computațional atâta timp cât funcția f este o funcție pseudo-aleatoare sigură. De aceea, este suficient să se demonstreze că există un simulator S astfel încât distribuția valorilor returnate în cazul H_1 nu poate fi diferențiată computațional de $SIM^{ORE}_{A,S,\mathcal{L}_f}(s)$

În cadrul simulatorului $S = (S_0, \ldots, S_q)$, S_0 inițializează un tabel pentru căutări, $T : [q] \times [n] \to \mathbb{Z}_M$, care mai apoi returnează $st_S = T$.

Pentru fiecare $t \in [q]$, după ce adversarul returnează o interogare m_t, este invocat algoritmul de simulare S_t, având la intrare $st_S = T$ și $\mathcal{L}_f(m_1, \ldots, m_t)$.

$\mathcal{L}_f(m_1, \ldots, m_t)$ conține valoare $1 \cdot (m_i < m_j)$ și $ind_{dif}(m_j, m_t)$ pentru toți $j \in [t-1]$, unde $ind_{dif}(m_j, m_t)$ este indexul primului bit care diferă în m_j și m_t.

Pentru fiecare $a \in [n]$ există 3 cazuri posibile:

- Există $j \in [t-1]$ astfel încât $ind_{dif}(m_i, m_j) > a$. Dacă există mai mulți j pentru care $ind_{dif}(m_i, m_j) > a$, se va lua în considerare cel mai mic j. Simulatorul setează $\bar{u}_a = T(j,a)$
- Pentru fiecare $l \in [t-1], ind_{dif}(m_l, m_t) \leq a$ și există un $j \in [t-1]$ pentru care $ind_{dif}(m_j, m_t) = a$. Dacă există mai mulți j care îndeplinesc condiția, va fi ales cel mai mic. Simulatorul setează $\bar{u}_a = T(j,a) - (1 - 2 \cdot (m_j < m_t)) \pmod M$
- Pentru fiecare $l \in [t-1], ind_{dif}(m_j, m_t) < a$. În acest caz simulatorul va genera $y \xleftarrow{R} \mathbb{Z}_M$ și setează $\bar{u}_a = y$

Pentru fiecare $a \in [n]$, simulatorul adaugă $(t,a) \mapsto \bar{u}_a$ în T. În final, simulatorul S_t returnează criptotextul $\bar{c}_t = (\bar{u}_1, \bar{u}_2, \ldots, \bar{u}_n)$ și actualizează starea $st_S = T$.

Se va demonstra că simulatorul $S = (S_0, S_1, \ldots, S_t)$ descrie în mod perfect distribuția din experimentul H_1. Fie $(c_1, \ldots, c_q)$ distribuția de probabilitate comună a criptotextelor

returnate de H_1 și $(\bar{c}_1, \ldots, \bar{c}_q)$ distribuția de probabilitate comună a criptotextelor returnate de simulator. Demonstrația utilizează inducția după q.

Cazul de bază, adică cel în care $q = 0$ este trivial.

Se presupune că $(c_1, \ldots, c_{t-1}) \equiv (\bar{c}_1, \ldots, \bar{c}_{t-1})$, unde $t \in [q]$. Se va demonstra că afirmația are loc pentru $t+1$.

În primul rând, pentru orice $j \in [t]$, se vor scrie $c_j = (u_{j,1}, u_{j,2}, \ldots, u_{j,n})$ și $\bar{c}_j = (\bar{u}_{j,1}, \bar{u}_{j,2}, \ldots, \bar{u}_{j,n})$, iar pentru fiecare $j \in [t]$, se notează cu $b_{j,a}$, cel de-al a-lea bit din m_j.

Pentru fiecare $a \in [n]$, există 3 cazuri:

- Există $j \in [t-1]$ astfel încât $ind_{dif}(m_j, m_t) > a$, iar dacă există mai mulți se va lua în considerare doar cel cu valoarea cea mai mică. Asta înseamnă că m_j și m_t au un prefix comun de dimensiune cel puțin a. Se notează cu $p \in \{0,1\}^{a-1}$ primii $a-1$ biți ai acestui prefix comun. Atunci, în H_1, va avea loc

  $$u_{t,a} = F(a, p||0^{n-a}) + b_{t,a} = u_{j,a}$$

  În cadrul simulării, $u_{t,a} = u_{j,a}$. Cum $j < t$, se deduce că $u_{t,a}$ și $\bar{u}_{t,a}$ sunt identic distribuite.

- Pentru orice $l \in [t-1], ind_{dif}(m_l, m_t) \leq a$ și există $j \in [t-1], ind_{dif}(m_j, m_t) = a$, iar dacă există mai multe valori posibile pentru j, este luată în considerare doar cea mai mică. De aici rezultă că m_j și m_t au același prefix $p \in \{0,1\}^{a-1}$ de dimensiune $a-1$. Apoi, în H_1,

  $$u_{t,a} = F(a, p||0^{n-a}) + b_{t,a} \pmod{M}$$

  În simulare

  $$\bar{u}_{t,a} = T(j,a) - (1 - 2 \cdot 1 \cdot (m_j < m_t)) = \bar{u}_{j,a} - (1 - 2 \cdot 1 \cdot (m_j < m_t)) \pmod{M}.$$

  În H_1, $u_{j,a} = F(a, p||0^{n-a}) + b_{j,a}$. Din presupunerea că $b_{j,a} \neq b_{t,a}$, se poate scrie $b_{t,a} = b_{j,a} - (1 - 2 \cdot 1 \cdot (m_j < m_t)))$. De aceea, în H_1 are loc

  $$u_{t,a} = F(a, p||0^{n-a+1}) + b_{t,a} = u_{j,a} - (1 - 2 \cdot (m_j < m_t)) \pmod{M}$$

Din ipoteza inductivă se știe că $u_{j,a}$ și $\bar{u}_{j,a}$ sunt identic distribuite și se ajunge la concluzia că $u_{t,a}$ și $\bar{u}_{t,a}$ sunt identic distribuite.

- Pentru fiecare $l \in [t-1], ind_{dif}(m_l, m_t) < a$. Fie $p \in \{0,1\}^{a-1}$ primii $a-1$ biți din m_t. În H_1, are loc

$$u_{t,a} = F(a, p||0^{n-a} + b_{t,a}) \pmod{M}$$

Se știe că mesajele $m_1, \ldots, m_{t-1}$ încep cu prefixul p. Din moment ce F este o funcție aleatoare, valoarea $F(a, p||0^{n-a})$ este aleasă uniform din $\mathbb{Z}_M$ și este independentă de celelalte criptotexte. Așadar, $u_{t,a}$ și $\bar{u}_{t,a}$ sunt identic distribuite.

În concluzie, pentru orice $a \in [n], u_{t,a} \equiv \bar{u}_{t,a}$. Din moment ce componentele fiecărui criptotext sunt construite în mod independent în H_1 și în simulare, este suficient să se demonstreze că c_t și $\bar{c}_t$ sunt identic distribuite. Validitatea afirmației decurge în urma inducției după t.

Schema de criptare care păstrează relația de ordine prezentată, produce pentru un plaintext, având dimensiunea de n biți, un criptotext de $\lceil n \cdot log_2 M \rceil$ biți. Pentru cazul în care $M = 3$, criptotextul va ocupa $\lceil n \cdot log_2 3 \rceil \approx 1.59 \cdot n$ biți.

4. Detalii de implementare

Partea practică a lucrării constă în implementarea, în limbajul Python, a două dintre protocoalele prezentate în capitolul al doilea și respectiv al treilea:

- Un protocol de criptare simetrică ce permite căutarea unui cuvânt în criptotext, propus în [1] și descris în capitolul *Căutarea unui cuvânt în criptotext*. De acum încolo acesta va fi denumit procolul lui Song.
- Un protocol de criptare ce dezvăluie ordinea, propus în [13] și prezentat în secțiunea *Criptare ce dezvăluie relația de ordine*. Acesta va fi numit protocolul ORE.

Protocolul lui Song

Scenariul are în vedere două obiecte: client și administrator. Clientul este cel care:

- deține o colecție de documente
- le criptează și le transmite administratorului
- face o interogare către administrator pentru documentele care conțin un anumit termen, fără a face termenul cunoscut administratorului

Administratorul este cel care:

- păstrează documentele în format criptat
- verifică dacă un anumit document conține cuvântul indicat de client, fără a afla despre ce cuvânt este vorba

- verifică pe rând fiecare document din colecție și le transmite clientului pe acelea care conțin termenul dorit

Protocolul se bazează pe: o funcție pseudo-aleatoare, o permutare pseudo-aleatoare și un generator pseudo-aleator de numere (a se vedea secțiunile *2.1. Criptare simetrică* și *1. Primitive criptografice*). Pe baza rezultatelor din [14], am ales ca funcție pseudo-aleatoare un cod de autentificare a mesajelor bazat pe funcții hash, pe scurt HMAC (engl: *Hash-based Message Authentication Code*). HMAC, așa cum apare [15], are următoarea definiție:

$$HMAC(k,m) = H((k' \oplus opad)||H((k' \oplus ipad)||m)),$$

unde

H este o funcție hash

k este cheia secretă

m este mesajul

k' este o altă cheie secretă derivată din cheia k

- apendând "0" până când lungimea acesteia atinge dimensiunea blocului de intrare
- apelând fucția de hash, având la intrare valoarea cheii k

$opad$ este un bloc hexadecimal de mărimea blocului de intrare de forma 0x5c5c5c...5c5c

$ipad$ este un bloc hexadecimal de mărimea blocului de intrare de forma 0x363636...3636

Am ales ca funcție de hash funcția SHA-256, așadar lungimea blocului de intrare în cazul de față va fi 256.

Pentru generarea de numere pseudo-aleatoare am utilizat metoda *getrandbits* din librăria *random* pentru Python. Aceasta implementează algoritmul *Mersenne Twister*, care

conform [16] se încadrează în categoria generatoarelor de numere pseudo-aleatoare. Ca permutare pseudo-aleatoare am ales standardul actual de criptare simetriă, adică AES(a se vedea secțiunile *1.3. Permutări de numere pseudo-aleatoare* și *1.5. Criptare simetrică*). Algoritmul de criptare AES a fost apelat în modul cu o cheie de 128 de biți. Modul CBC presupune:

- Se execută XOR între primul bloc de plaintext și un vector inițializare *IV* pe 128 de biți, iar apoi se realizează criptarea bloc standard, utilizând cheia *k*, dată la intrare, respectând pașii descriși în secțiunea *1.5. Criptare simetrică*.
- Pentru fiecare din următoarele blocuri, înainte de criptare, se execută XOR între blocul criptat anterior și blocul curent de plaintext.

Clasa *Client* deține următoarele proceduri esențiale:

- *enc*
  intrare: cuvântul *W* și sămânța *seed*

  1. se apelează metoda *simetric_enc*, care realizează o criptare simetrică AES pentru un cuvânt reprezentat sub formă binară
  2. se împarte cuvântul în *L* și *R*
  3. se generează folosind *seed* o secvență aleatoare de biți *S*
  4. se calculează cheia de criptare curentă utilizând valoarea funcției aleatoare HMAC calculată având la intrare *MasterKey* și *L*
  5. se apelează funcția pseudo-aleatoare având la intrare cheia *key* obținută la pasul anterior și *S*
  6. se concatenează *S* cu secvența obținută la pasul anterior și se obține secvența *SFk*
  7. se aplică XOR peste *SFk* și secvența obișnuită în urma criptării simetrice de la primul punct

  ieșire: cuvânt criptat *C*

- *dec*
  intrare: criptotextul C și sămânța *seed*
  Decriptarea urmează în ordine inversă pașii din algorimul de criptare *enc*.
  ieșire: cuvântul W

- *send_files*
  intrare: numele unui administrator, o listă de nume de fișiere
  Se preia fiecare fișier în parte, se criptează pe rând fiecare cuvânt și se trimit administratorului.

- Trapdoor
  intrare: un cuvânt W
  ieșire: tuplul (X,k), unde X este criptotextul cuvântului W, iar k este cheia necesară verificării.

La fiecare pas, adică pentru fiecare cuvânt dintr-un fișier am considerat ca *seed* cuvântul anterior.

Cea mai importantă metodă a clasei *Admin* este $verify$. Aceasta are la intrare un criptotext C și un tuplu (X,k), reprezentând criptarea standard a cuvântului dorit și cheia k corespunzătoare acestuia. Această metodă este de a verifica dacă criptotextul C corespunde cuvântului dorit W, testând dacă se poate obține din X, folosind k, criptotextul C.

Protocolul ORE

Rolul acestui protocol este de a crea criptotexte care pot fi comparate între ele și dezvăluie relația de ordine a plaintextelor aferente. Clasa *ORE* conține următoarele metode:

- *enc*:
  intrare: un plaintext m
  Pentru fiecare bit se calculează:

$$u_i = f(k,(i,b_1b_2\ldots b_{i-1}||0^{n-i})) + b_i \pmod M$$

(a se vedea secțiunea *3.3. Criptare care dezvăluie relația de ordine*). La finalul șirului astfel obținut se concatenează mesajul m criptat cu AES, care va servi ulterior procesului de decriptare.

ieșire: criptotextul C

- *comparare*:
  intrare: două criptotexte C și C'
  Se preiau u și u' din C și C'
  Fie $i = ind_{dif}(C_1, C_2)$
  ieșire: $\begin{cases} True, & \text{dacă nu există un astfel de index sau există, dar } u'_i \neq u_i + 1 \ (mod\ M) \\ False, & \text{altfel.} \end{cases}$

Concluzii

Cum prima schemă de criptare care permite căutarea unui cuvânt în criptotext a fost propusă în anul 2000 ([8]), iar prima schemă de criptare care păstrează ordinea în anul 2004 ([11]), se poate afirma că domeniul căutării în date criptate este încă efervescent.

Nevoia de a căuta informații în datele criptate duce la punerea în discuție a tot mai multor scenarii, uzuale în situații reale, precum:

- Cazul în care clientul introduce un termen greșit sau incomplet pentru care dorește o căutare
- Cazul în care clientul dorește să acorde altor utilizatori drepturi de a face căutări asupra datelor proprii
- Cazul în care clientul dorește căutarea în funcție de mai multe cuvinte

Un alt inconvenient este legat de raportul dintre eficiență și securitate. S-a constatat că dintre schemele propuse până în prezent, cele care asigură cel mai ridicat grad de securitate sunt și cele mai ineficiente, necesitând resurse suplimentare precum timpul de comunicare client-server din cazul criptării mutabile (a se consulta secțiunea 3.2.). De asemenea, nivelul de securitate în situații practice, al diverselor scheme propuse nu este bine definit și de aceea, se recomandă precauție înainte de utilizarea acestora. Unele dintre tehnici se bazează pe presupoziții stricte legate de primitivele utilizate sau de

Întrucât căutarea în date criptate răspunde unor necesități de actualitate, aceasta constituie o ramură în plină dezvoltare a criptografiei.

Bibliografie

[1] M. Luby, C. Rackoff - *How to construct pseudorandom permutations and pseudorandom functions*, SIAM J. Comput., vol. 17, 373-386, 1988.

[2] A. Menezes, P. van Oorschot, S. Vanstone - *Handbook of applied cryptography*, 283-288, iunie 1996.

[3] Dan Boneh, CS355 class, https://crypto.stanford.edu/pbc/notes/crypto, 2002

[4] O.Goldreich, S. Golldwasser, S. Micali - *How to construct random functions*, J. ACM 33, 792–807, 1986.

[5] Rajeev Sobti, G. Geetha, - *Cryptographic Hash Functions: A Review*, IJCSI International Journal of Computer Science Issues, Vol. 9, Issue2, No 2, 19, martie 2012.

[6] R. Rivest - *The MD5 Message-Digest Algorithm*, RFC 1321, 21, aprilie 1992.

[7] B. Preneel - *Analysis and Design of Cryptographic Hash Functions*, 192-196, februarie 2003.

[8] D. Song, D. Wagner, A. Perrig - *Practical techniques for searches on encrypted data*, Proc. of IEEE Symposium on Security and Privacy'00, 12, 2000.

[9] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky - *Searchable symmetric encryption: Improved definitions and efficient constructions*. Cryptology ePrint archive, 10, iunie 2006.

[10] D. Boneh, G. Di Crescenzo, R. Ostrovsky, G. Persiano - *Public key encryption with keyword search*. Eurocrypt, Springer, 12, 2004

[11] R. Agrawal, J. Kiernan, R. Srikant, Yirong Xu - *Order preserving encryption for numeric data*. SIGMOD, 563-574, 2004.

[12] R. A. Popa, F. H. Li, N Zeldovich - *An ideal-security protocol for order-preserving encoding*. IEEE Symposium on Security and Privacy, 463-477, 2013.

[13] N. Chenette, K. Lewi, S.A. Weis, D.J. Wu - *Practical Order-Revealing Encryption with Limited Leakage*, FSE, 27, 2016

[14] M Bellare - *New Proofs for NMAC and HMAC: Security without Collision Resistance*. Cryptology, 30, 2015

[15] H. Krawczyk,M. Bellare, R. Canetti - *HMAC: Keyed-Hashing for Message Authentication*. RFC 2104, 11, februarie 1997

[16] M Matsumoto, T Nishimura - *Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator*, ACM Transactions on Modeling and Computer Simulations, 26, 1998