Securitatea Sistemelor Informatice

- Curs 8.2 -
Aplicații ale funcțiilor hash

Adela Georgescu

Facultatea de Matematică și Informatică
Universitatea din București
Anul universitar 2022-2023, semestrul I

Atac pentru găsirea de coliziuni

- Considerăm funcții hash

$$H : \{0,1\}^* \to \{0,1\}^n$$

Atac pentru găsirea de coliziuni

- Considerăm funcții hash

$$H : \{0,1\}^* \to \{0,1\}^n$$

- Rețineți! Cel mai bun atac generic pentru găsirea de coliziuni ne spune că avem nevoie de functii hash cu output-ul pe $2n$ biți pentru securitate împotriva adversarilor care rulează în timp 2^n...

Atac pentru găsirea de coliziuni

- Considerăm funcții hash

$$H : \{0,1\}^* \to \{0,1\}^n$$

- Rețineți! Cel mai bun atac generic pentru găsirea de coliziuni ne spune că avem nevoie de functii hash cu output-ul pe $2n$ biți pentru securitate împotriva adversarilor care rulează în timp 2^n...

- ... spre deosebire de sistemele de criptare bloc, unde avem nevoie de n biți pentru securitate împotriva adversarilor care rulează în timp 2^n

Atac pentru găsirea de coliziuni

- Considerăm funcții hash

$$H : \{0,1\}^* \to \{0,1\}^n$$

- Rețineți! Cel mai bun atac generic pentru găsirea de coliziuni ne spune că avem nevoie de functii hash cu output-ul pe $2n$ biți pentru securitate împotriva adversarilor care rulează în timp 2^n...

- ... spre deosebire de sistemele de criptare bloc, unde avem nevoie de n biți pentru securitate împotriva adversarilor care rulează în timp 2^n

Atac pentru găsirea de coliziuni

- Considerăm funcții hash

$$H : \{0,1\}^* \to \{0,1\}^n$$

- Rețineți! Cel mai bun atac generic pentru găsirea de coliziuni ne spune că avem nevoie de functii hash cu output-ul pe $2n$ biți pentru securitate împotriva adversarilor care rulează în timp 2^n...

- ... spre deosebire de sistemele de criptare bloc, unde avem nevoie de n biți pentru securitate împotriva adversarilor care rulează în timp 2^n

- Exemplu. Dacă dorim ca găsirea de coliziuni să necesite timp 2^{128} atunci trebuie să alegem functii hash cu output-ul pe 256 biți

Alte aplicații ale funcțiilor hash

- Am vazut ca functiile hash pot fi folosite pentru a construi MAC-uri de lungime variabila (în HMAC)

$$x \to H(x) \to Mac_k(H(x))$$

pentru MAC de lungime fixă

Alte aplicații ale funcțiilor hash

- Am vazut ca functiile hash pot fi folosite pentru a construi MAC-uri de lungime variabila (în HMAC)

$$x \to H(x) \to Mac_k(H(x))$$

pentru MAC de lungime fixă

- Functiile hash au si multe alte aplicatii atat in criptografie cat si in securitate

Alte aplicații ale funcțiilor hash

- Am vazut ca functiile hash pot fi folosite pentru a construi MAC-uri de lungime variabila (în HMAC)

$$x \rightarrow H(x) \rightarrow Mac_k(H(x))$$

pentru MAC de lungime fixă

- Functiile hash au si multe alte aplicatii atat in criptografie cat si in securitate
- Pentru un fișier x, $H(x)$ poate servi ca identificator unic (amprentă) - existența unui fișier diferit cu același hash implică gasirea de coliziuni in H).

Alte aplicații ale funcțiilor hash

- Am vazut ca functiile hash pot fi folosite pentru a construi MAC-uri de lungime variabila (în HMAC)

$$x \to H(x) \to Mac_k(H(x))$$

pentru MAC de lungime fixă

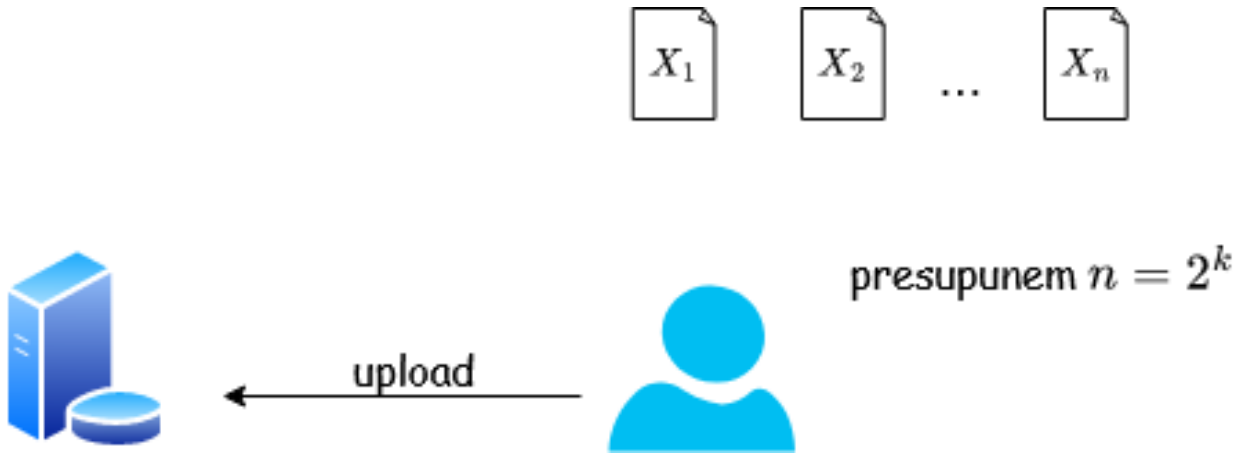
- Functiile hash au si multe alte aplicatii atat in criptografie cat si in securitate
- Pentru un fișier x, $H(x)$ poate servi ca identificator unic (amprentă) - existența unui fișier diferit cu același hash implică gasirea de coliziuni in H).
- Prezentam cateva aplicatii care decurg de aici

Alte aplicații ale funcțiilor hash

- *Amprentarea virusilor*
  - scanner-ele de viruși pastreaza hash-urile virusilor cunoscuti
  - verificarea fisierelor potential malitioase se face comparand hash-ul lor cu hash-ul virusilor cunoscuti.
- *Deduplicarea datelor* - stocarea in cloud partajata de mai multi utilizatori - se verifica daca hash-ul unui fisier nou (de incarcat) exista deja in cloud, caz in care se adauga doar un pointer la fisierul respectiv

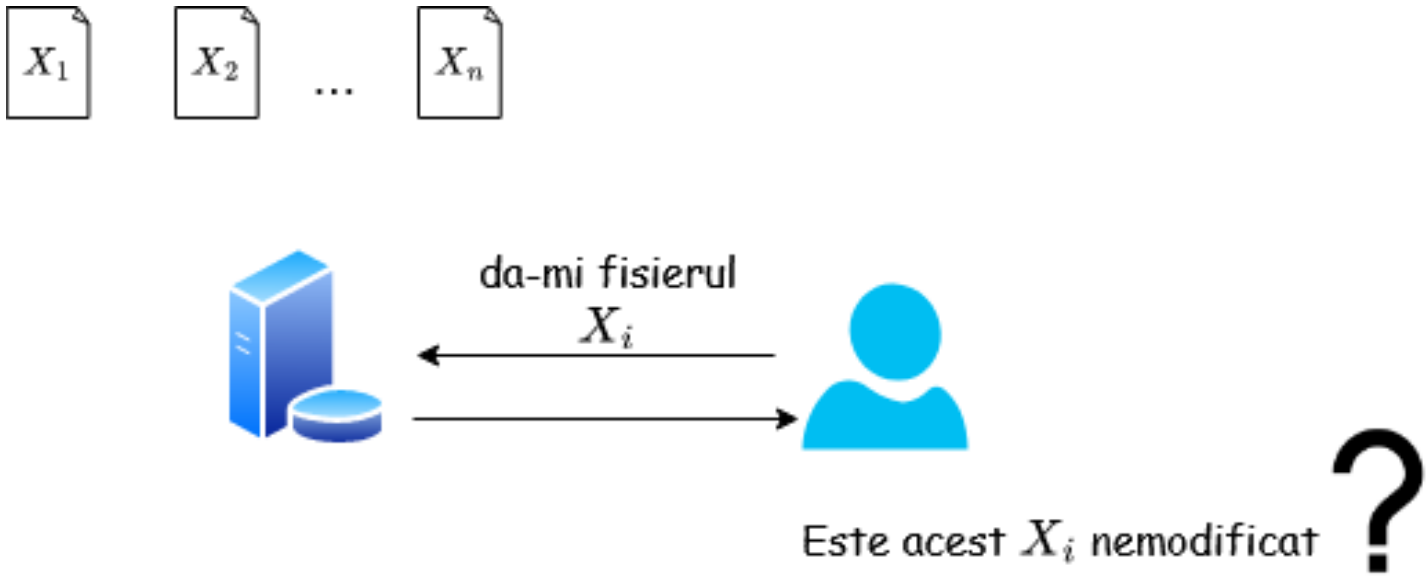
Arbori Merkle

- Un client incarca mai multe fisiere pe server ...



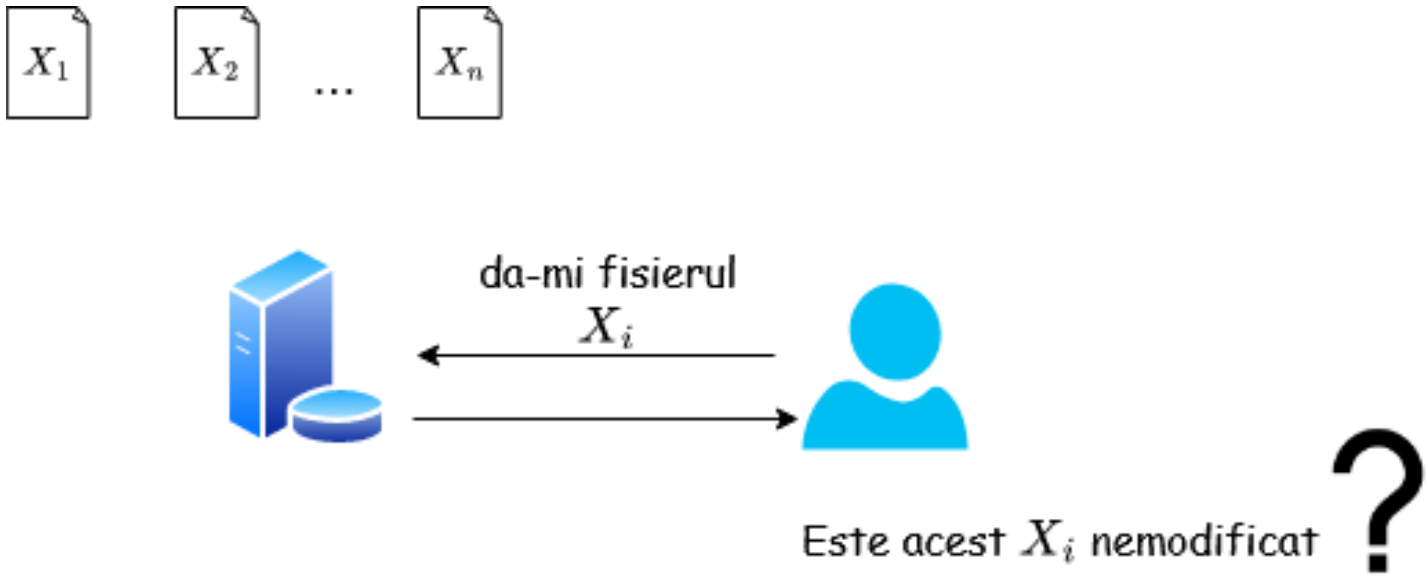
Arbori Merkle

- ... si doreste ca atunci cand le recupereaza de pe server, sa verifice ca nu au fost modificate



Arbori Merkle

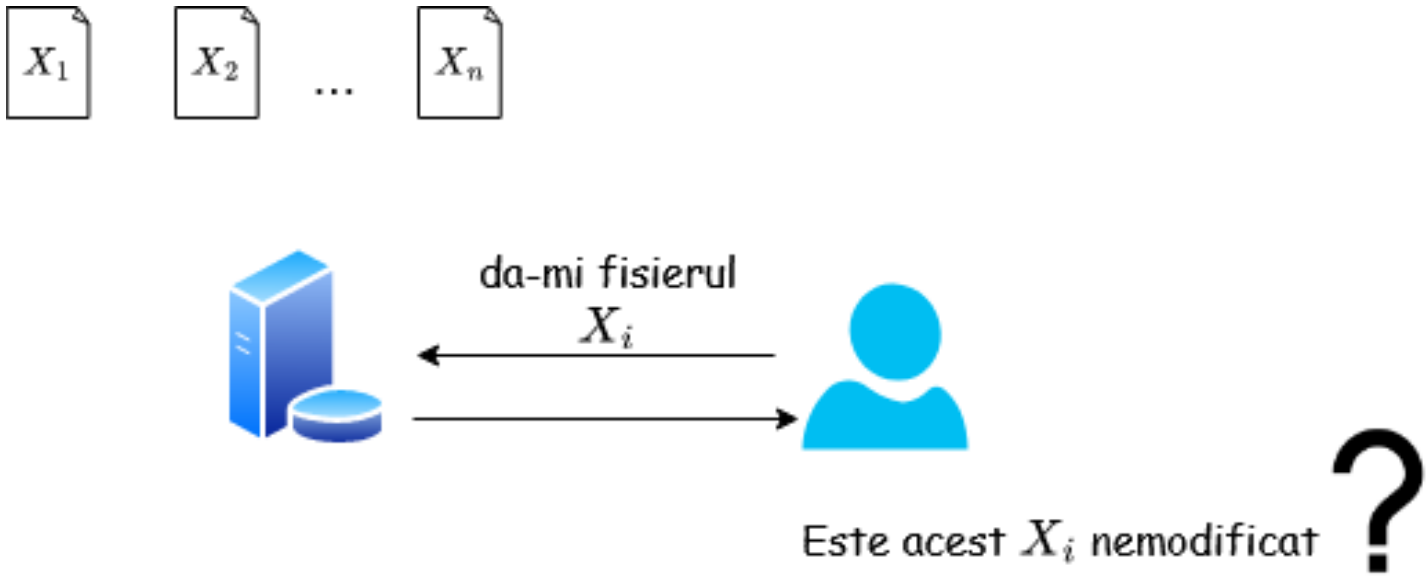
- ... si doreste ca atunci cand le recupereaza de pe server, sa verifice ca nu au fost modificate



1. Prima solutie:

Arbori Merkle

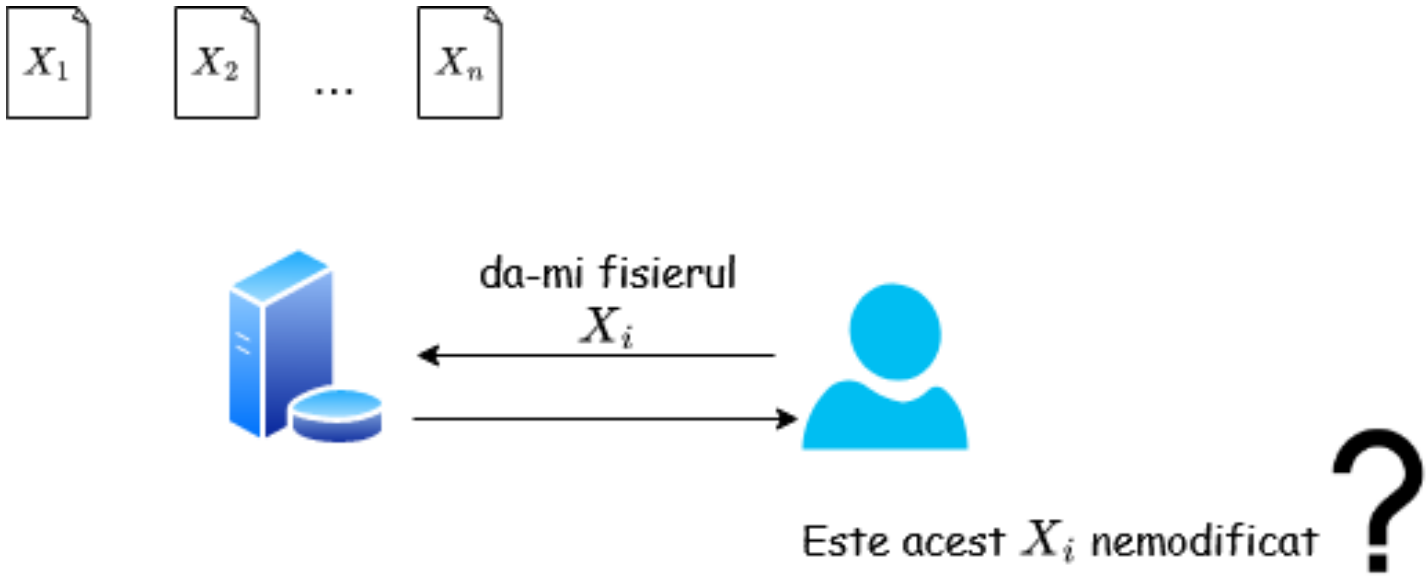
- ... si doreste ca atunci cand le recupereaza de pe server, sa verifice ca nu au fost modificate



1. Prima solutie:
   - clientul stochează local $h_1 = H(x_1), ..., h_n = H(x_n)$ și verifică dacă $h_i = H(x_i')$ pentru fiecare fișier x_i' recuperat

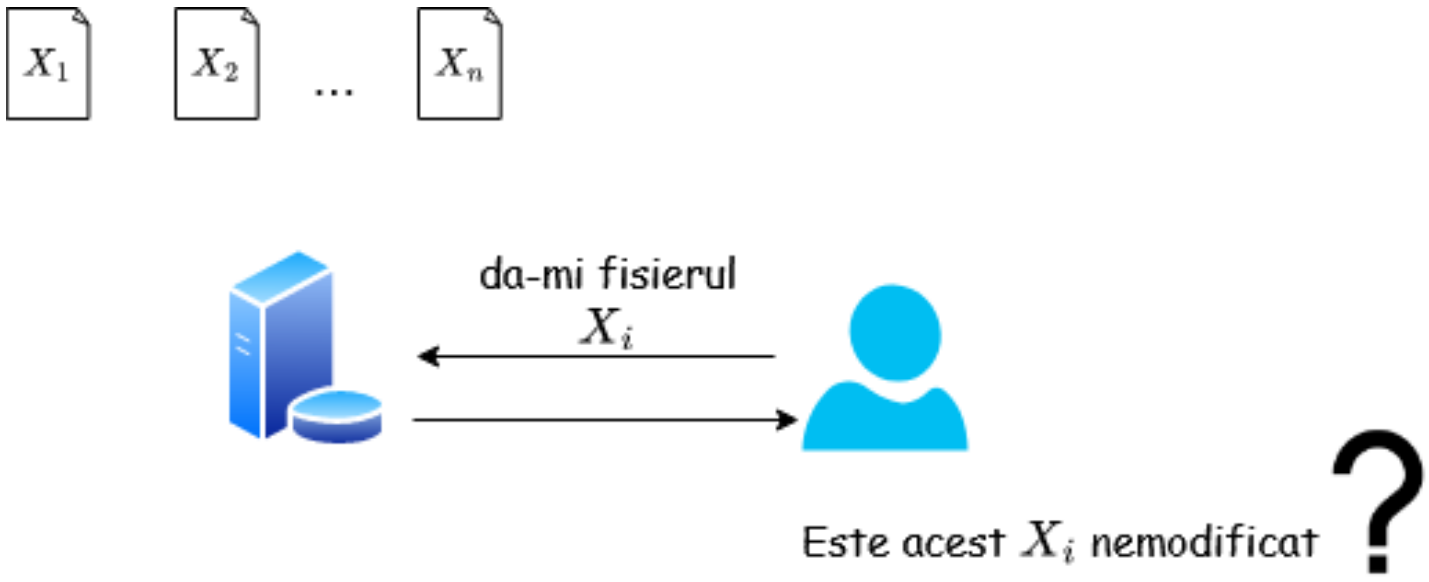
Arbori Merkle

- ... si doreste ca atunci cand le recupereaza de pe server, sa verifice ca nu au fost modificate

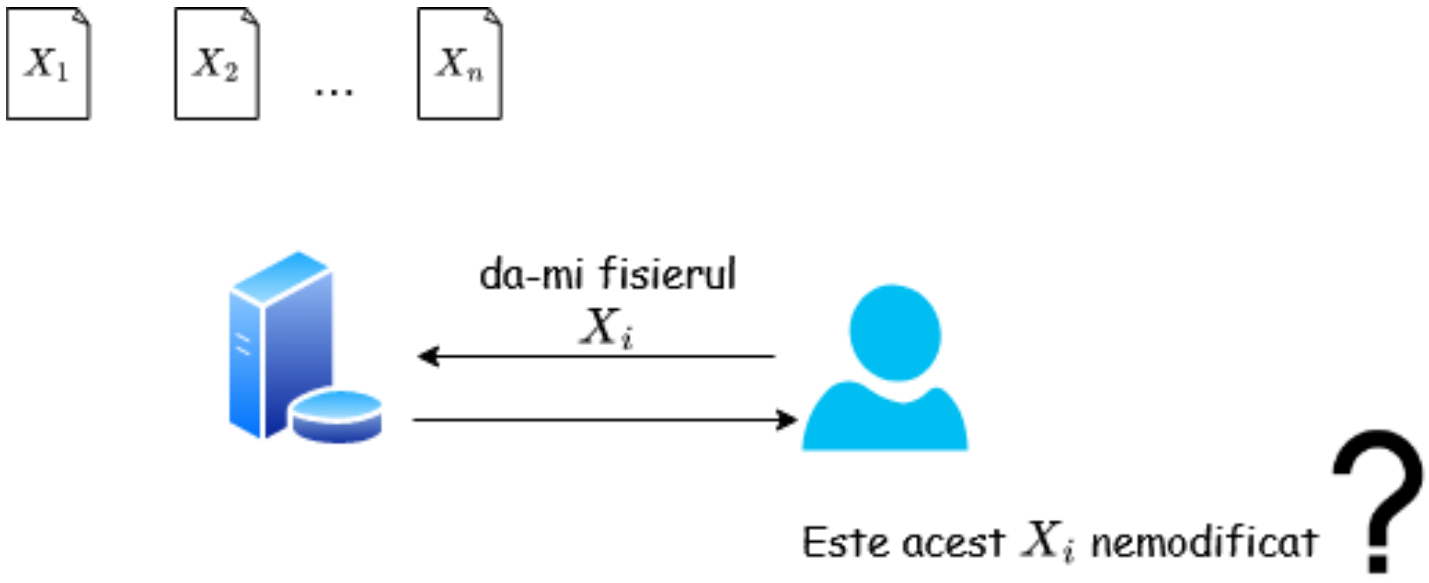


1. Prima solutie:
   - clientul stochează local $h_1 = H(x_1), ..., h_n = H(x_n)$ și verifică dacă $h_i = H(x_i')$ pentru fiecare fișier x_i' recuperat
   - Dezavantaj: spațiul necesar stocării crește liniar în n

Arbori Merkle

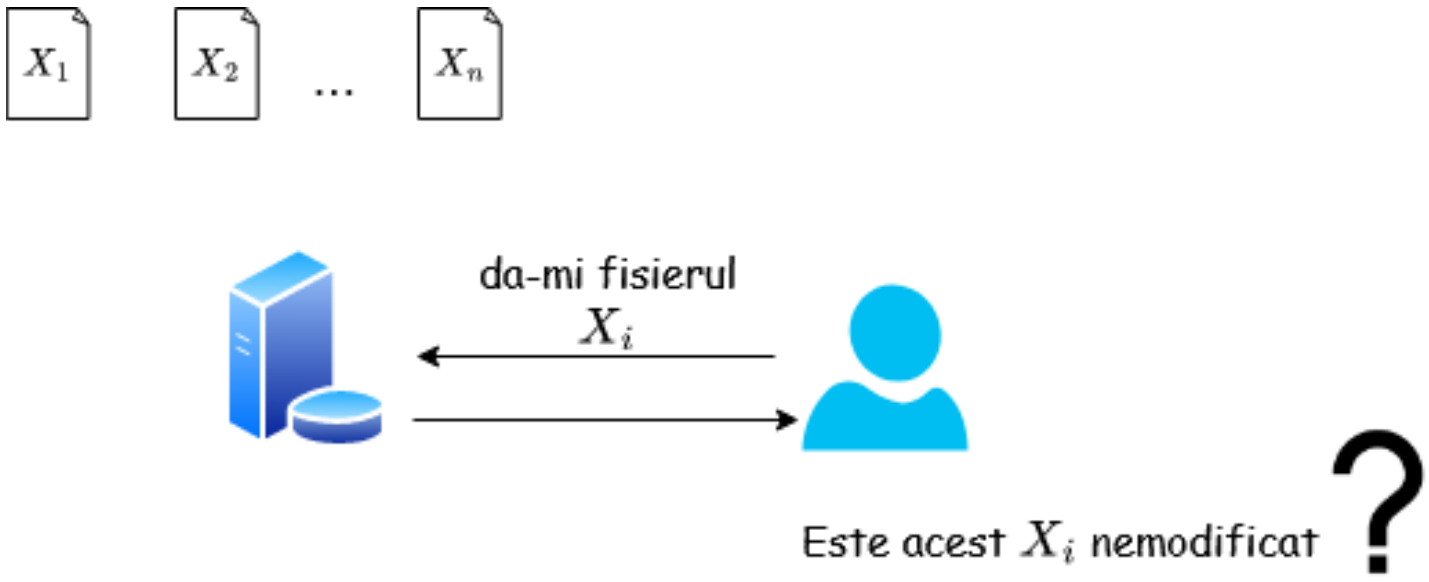


Arbori Merkle



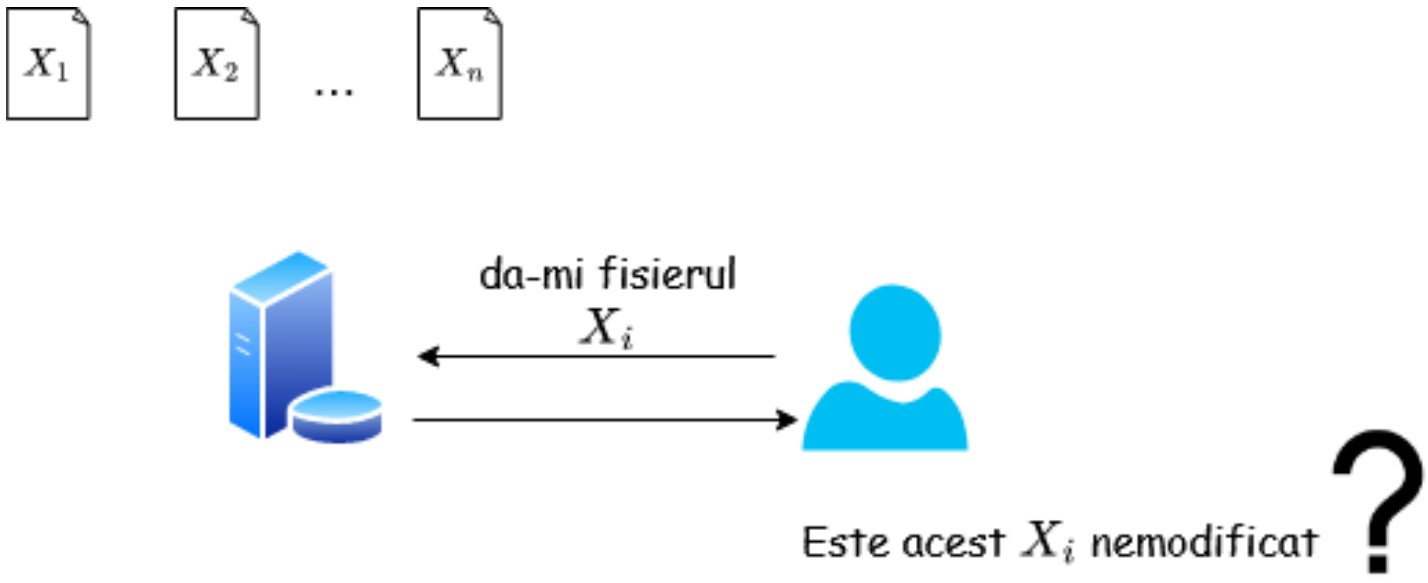
2. A doua soluție

Arbori Merkle



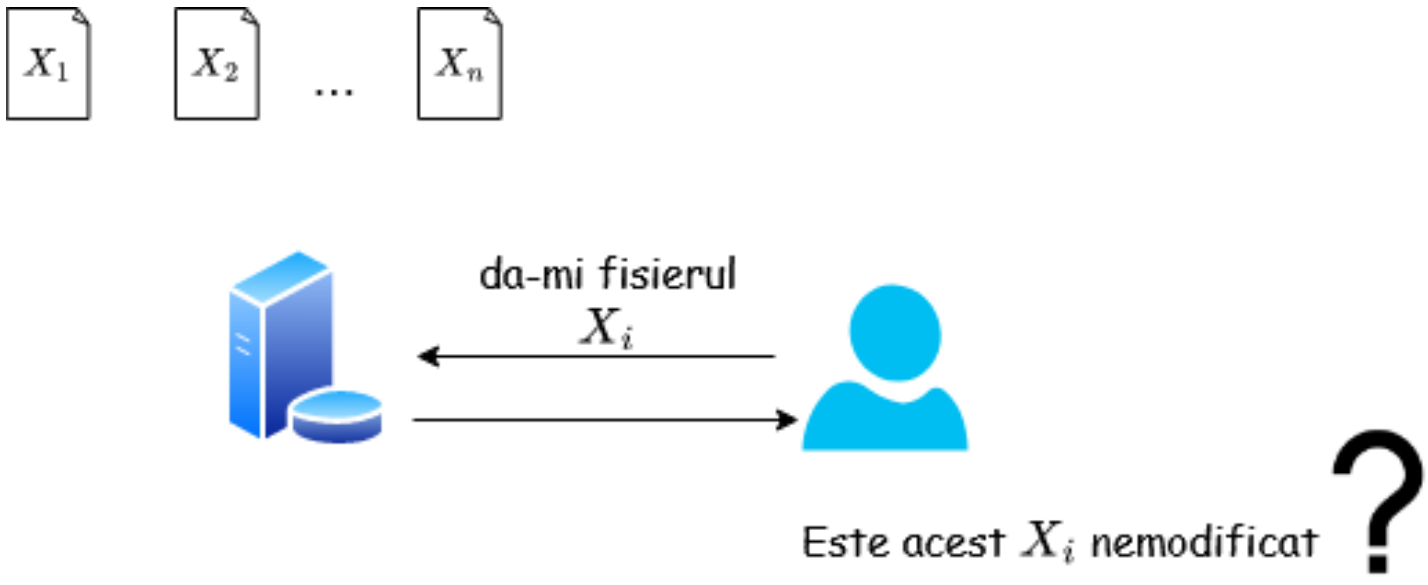
2. A doua soluție
   - clientul stocheaza local un singur hash $h = H(x_1, ..., x_n)$

Arbori Merkle



2. A doua soluție
   - clientul stocheaza local un singur hash $h = H(x_1, ..., x_n)$
   - Dezavantaj: pentru a verifica x_i, clientul trebuie să recupereze toate fișierele $x_1, ..., x_n$

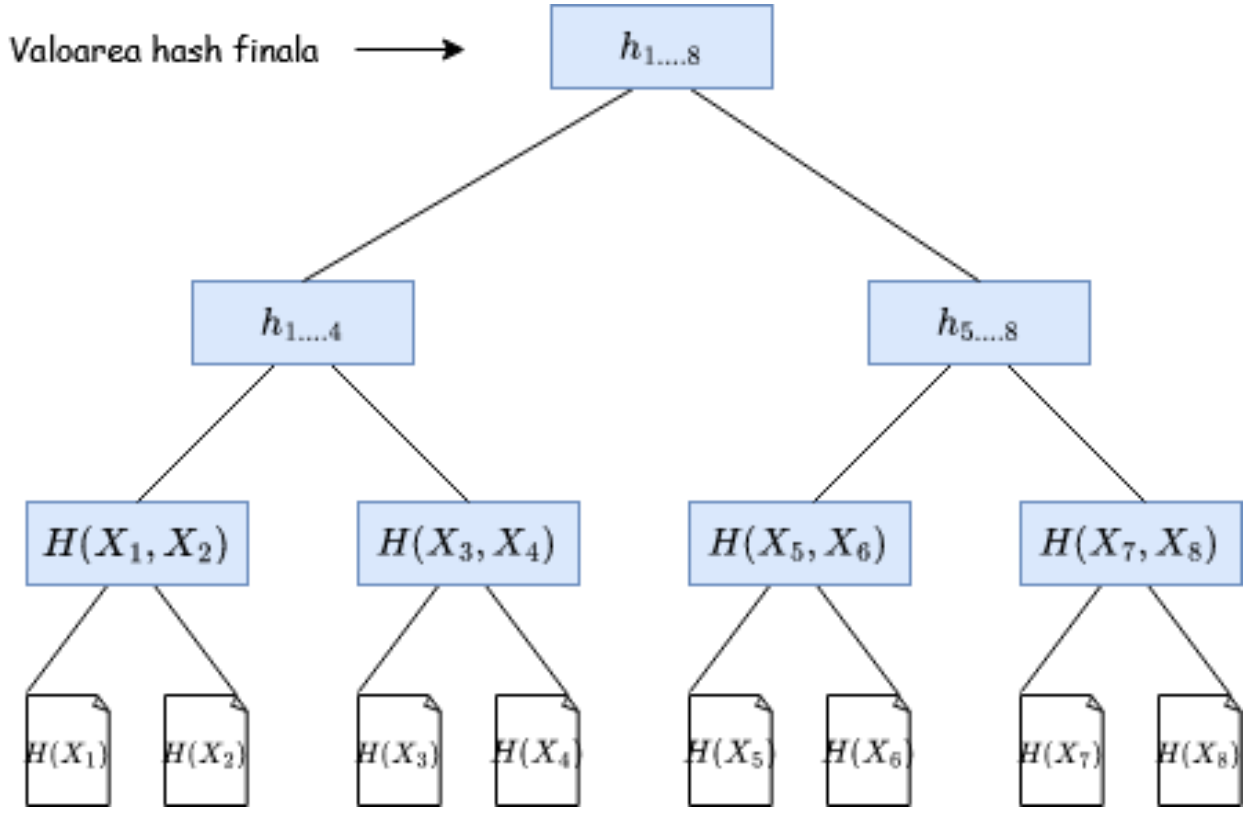
Arbori Merkle



2. A doua soluție
   - clientul stocheaza local un singur hash $h = H(x_1, ..., x_n)$
   - Dezavantaj: pentru a verifica x_i, clientul trebuie să recupereze toate fișierele $x_1, ..., x_n$
   - Soluție: arborii Merkle oferă un compromis între cele două soluții de mai sus

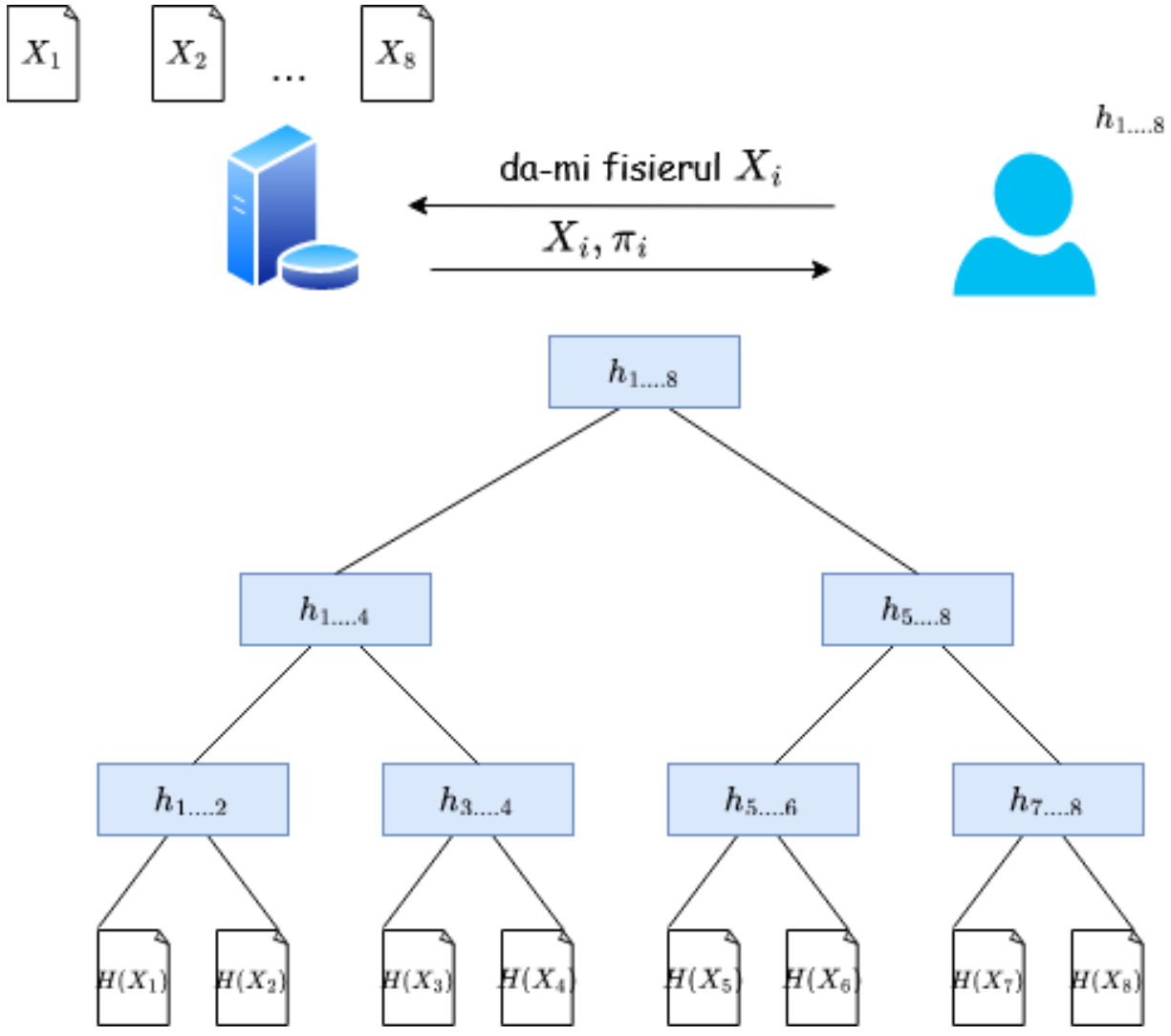
Arbori Merkle

- H - funcție hash rezistentă la coliziuni

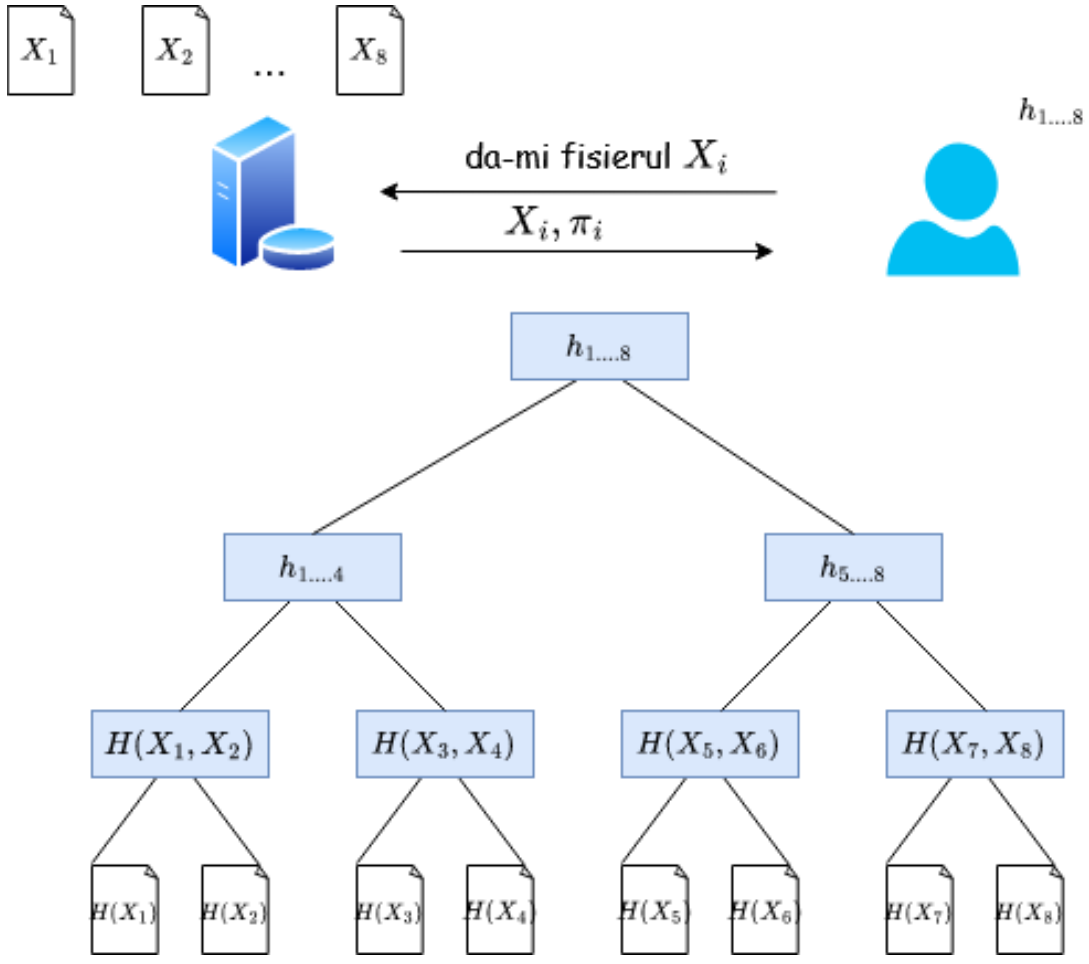


- Este o altenativa la constructia Merkle-Damgard pentru functii hash de lungime arbitrara

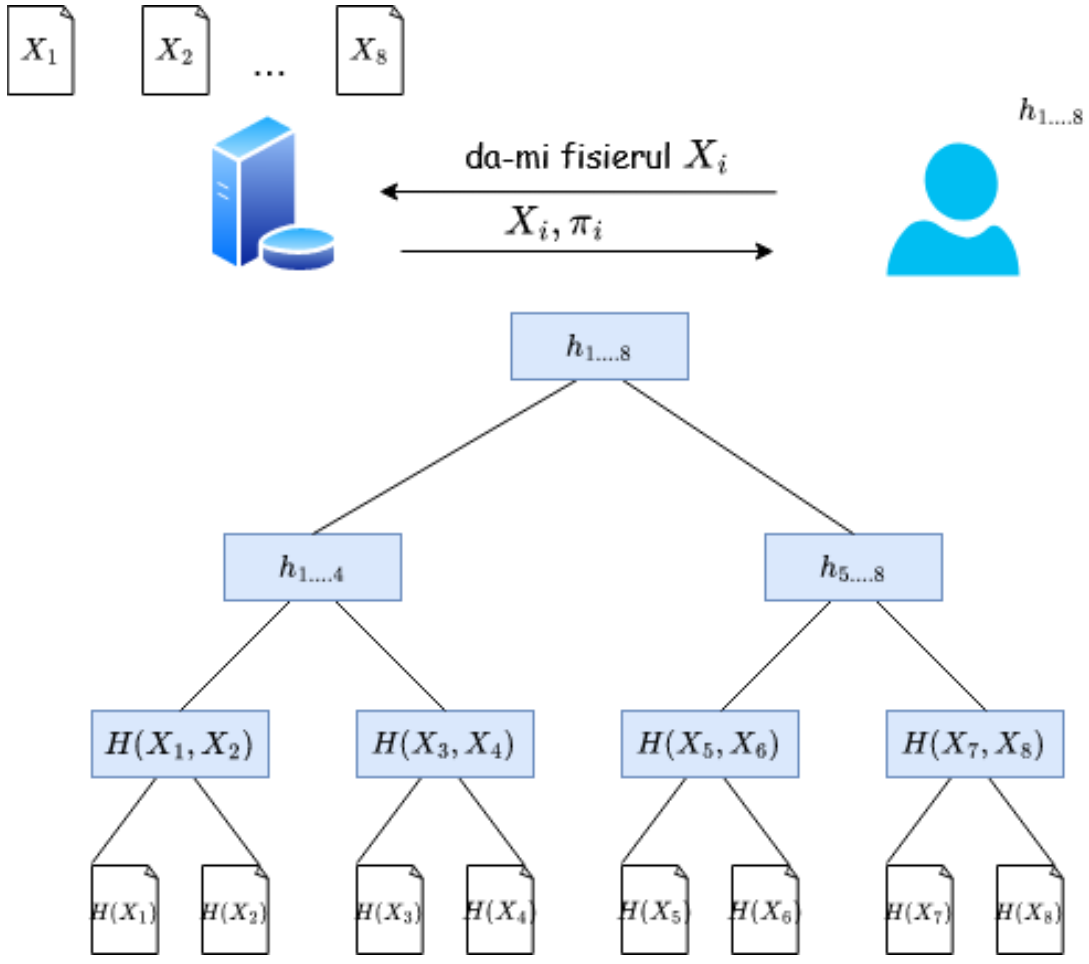
Arbori Merkle - stocarea fișierelor în cloud



Arbori Merkle - stocarea fișierelor în cloud

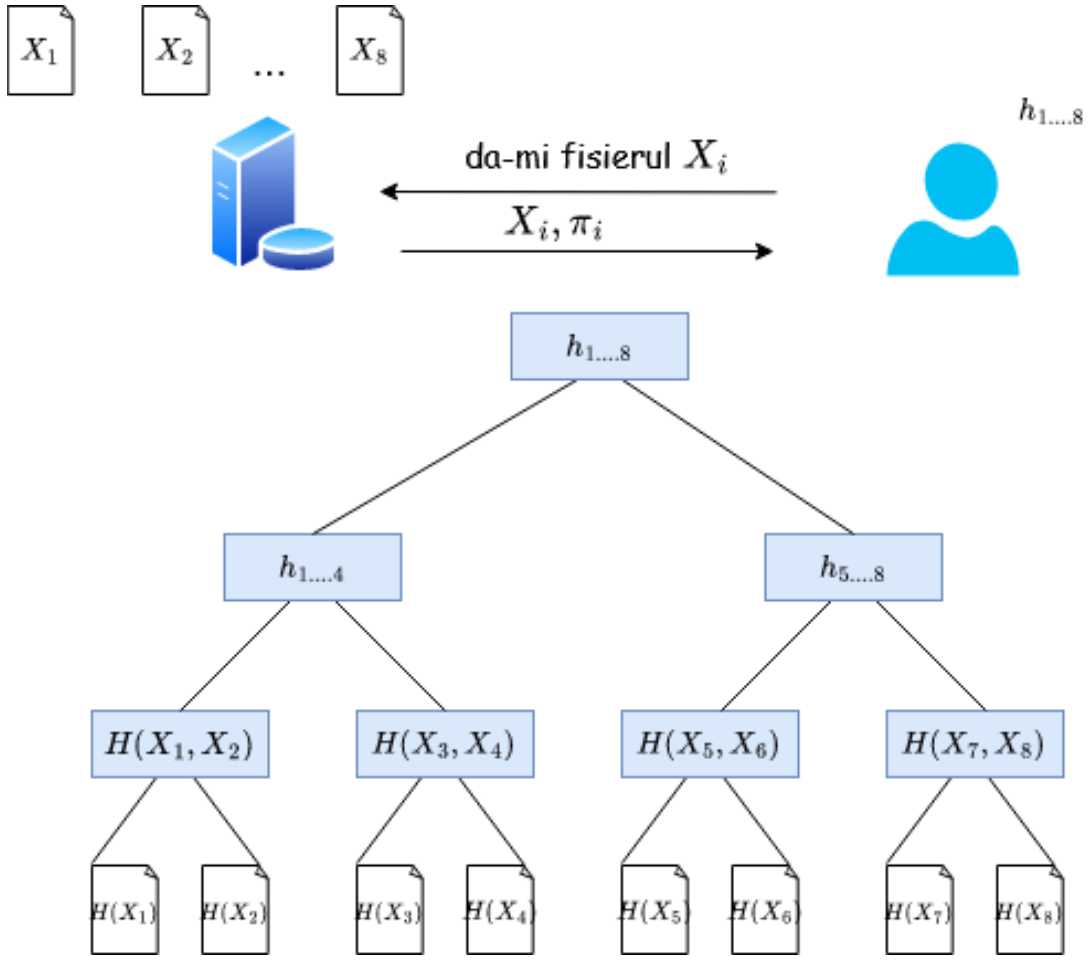


Arbori Merkle - stocarea fișierelor în cloud



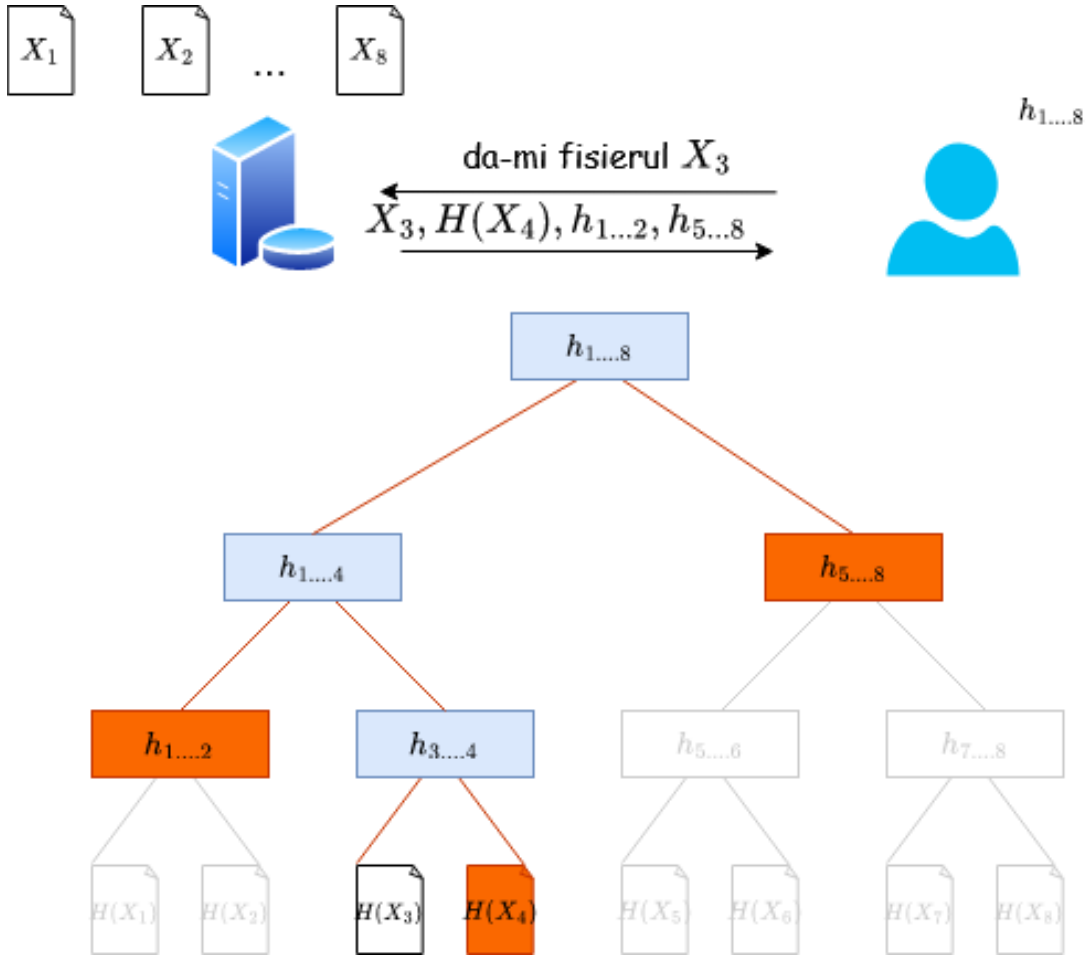
- π_i - conține nodurile adiacente drumului de la X_i la rădăcină

Arbori Merkle - stocarea fișierelor în cloud

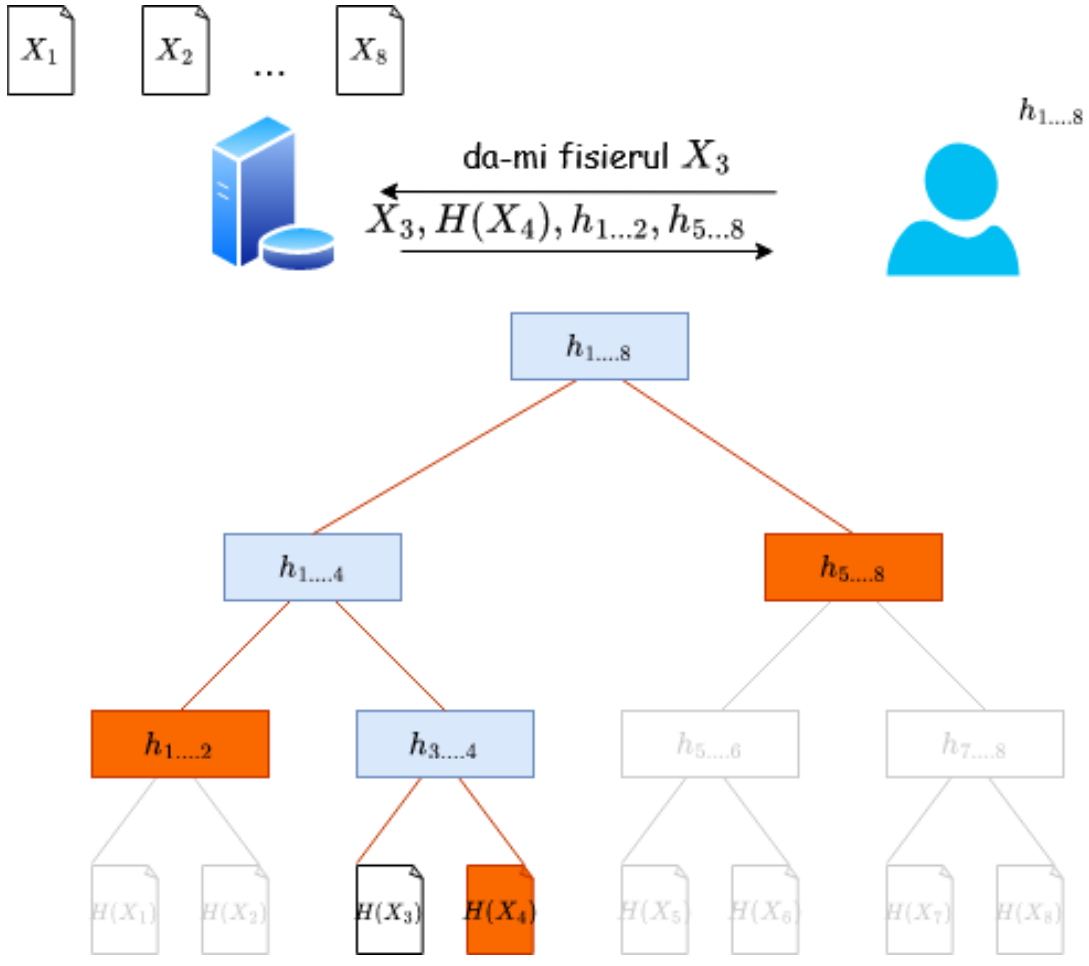


- π_i - conține nodurile adiacente drumului de la X_i la rădăcină
- pe baza lui π_i, user-ul poate re-calcula valoarea rădăcinii pentru a verifica dacă este aceeași cu valoarea stocată de el

Arbori Merkle - stocarea fișierelor în cloud



Arbori Merkle - stocarea fișierelor în cloud



- user-ul calculează $H(X_3)$, apoi $h'_{3...4} = H(H(X_3), H(X_4))$, $h'_{1...4}$ si $h'_{1...8}$ si verifica daca $h'_{1...8} = h_{1...8}$

Stocarea parolelor

- Cea mai utilizată metodă de autentificare este bazată pe nume de utilizator și parolă;

Stocarea parolelor

- Cea mai utilizată metodă de autentificare este bazată pe nume de utilizator și parolă;
- Metoda presupune o etapă de înregistrare, în care utilizatorul alege un *username* și o *parolă* (*pwd*);

Stocarea parolelor

- Cea mai utilizată metodă de autentificare este bazată pe nume de utilizator și parolă;
- Metoda presupune o etapă de înregistrare, în care utilizatorul alege un *username* și o *parolă* (*pwd*);
- Ulterior, la fiecare logare, utilizatorul introduce cele 2 valori;

Stocarea parolelor

- Cea mai utilizată metodă de autentificare este bazată pe nume de utilizator și parolă;
- Metoda presupune o etapă de înregistrare, în care utilizatorul alege un *username* și o *parolă* (*pwd*);
- Ulterior, la fiecare logare, utilizatorul introduce cele 2 valori;
- Dacă *username* se regăsește în lista de utilizatori înregistrați și parola introdusă este corectă atunci autentificarea se realizează cu succes;

Stocarea parolelor

- Cea mai utilizată metodă de autentificare este bazată pe nume de utilizator și parolă;
- Metoda presupune o etapă de înregistrare, în care utilizatorul alege un *username* și o *parolă* (*pwd*);
- Ulterior, la fiecare logare, utilizatorul introduce cele 2 valori;
- Dacă *username* se regăsește în lista de utilizatori înregistrați și parola introdusă este corectă atunci autentificarea se realizează cu succes;
- În caz contrar, autentificarea eșuază.

Stocarea parolelor

- O greșeală frecventă de implementare o reprezintă afișarea unor mesaje de tipul:

Nume de utilizator inexistent
Parolă greșită!

Stocarea parolelor

- O greșeală frecventă de implementare o reprezintă afișarea unor mesaje de tipul:

  Nume de utilizator inexistent
  Parolă greșită!

- Întrebare: De ce nu este indicată utilizarea unor astfel de mesaje de eroare?

Stocarea parolelor

- O greșeală frecventă de implementare o reprezintă afișarea unor mesaje de tipul:

  Nume de utilizator inexistent
  Parolă greșită!

- Întrebare: De ce nu este indicată utilizarea unor astfel de mesaje de eroare?

- Răspuns: Pentru că oferă informații suplimentare adversarului!

Stocarea parolelor

- O greșeală frecventă de implementare o reprezintă afișarea unor mesaje de tipul:

  Nume de utilizator inexistent
  Parolă greșită!

- Întrebare: De ce nu este indicată utilizarea unor astfel de mesaje de eroare?

- Răspuns: Pentru că oferă informații suplimentare adversarului!

- Corect este să se întoarcă un mesaj de eroare generic de tipul:

  Nume de utilizator sau parolă incorecte!

Stocarea parolelor

- O greșeală majoră este stocarea parolelor în clar!

Stocarea parolelor

- O greșeală majoră este stocarea parolelor în clar!
- Întrebare: De ce parolele NU trebuie stocate în clar?

Stocarea parolelor

- O greșeală majoră este stocarea parolelor în clar!
- Întrebare: De ce parolele NU trebuie stocate în clar?
- Răspuns: Pentru că dacă adversarul capătă acces la fisierul de parole atunci află direct parolele tuturor utilizatorilor!

Stocarea parolelor

- O greșeală majoră este stocarea parolelor în clar!
- Întrebare: De ce parolele NU trebuie stocate în clar?
- Răspuns: Pentru că dacă adversarul capătă acces la fisierul de parole atunci află direct parolele tuturor utilizatorilor!
- Pentru stocarea parolelor se utilizează funcțiile hash;

Stocarea parolelor

- O greșeală majoră este stocarea parolelor în clar!
- Întrebare: De ce parolele NU trebuie stocate în clar?
- Răspuns: Pentru că dacă adversarul capătă acces la fisierul de parole atunci află direct parolele tuturor utilizatorilor!
- Pentru stocarea parolelor se utilizează funcțiile hash;
- În fișierul de parole (sau baza de date) se stochează, pentru fiecare utilizator perechi de forma:

$$(username, H(pwd))$$

Stocarea parolelor

- Pentru autentificare, utilizatorul introduce numele de utilizator *username* și parola *pwd*;

Stocarea parolelor

- Pentru autentificare, utilizatorul introduce numele de utilizator *username* și parola *pwd*;
- Sistemul de autentificare calculează $H(pwd)$ și se verifică dacă valoarea obținută este stocată în fișierul de parole pentru utilizatorul indicat prin *username*;

Stocarea parolelor

- Pentru autentificare, utilizatorul introduce numele de utilizator *username* și parola *pwd*;
- Sistemul de autentificare calculează $H(pwd)$ și se verifică dacă valoarea obținută este stocată în fișierul de parole pentru utilizatorul indicat prin *username*;
- Dacă da, atunci autentificarea se realizează cu succes; în caz contrar, autentificarea eșuază;

Stocarea parolelor

- Pentru autentificare, utilizatorul introduce numele de utilizator *username* și parola *pwd*;
- Sistemul de autentificare calculează $H(pwd)$ și se verifică dacă valoarea obținută este stocată în fișierul de parole pentru utilizatorul indicat prin *username*;
- Dacă da, atunci autentificarea se realizează cu succes; în caz contrar, autentificarea eșuază;
- Funcțiile hash sunt funcții **one-way**: cunoscând $H(pwd)$ nu se poate determina *pwd*;

Stocarea parolelor

- Pentru autentificare, utilizatorul introduce numele de utilizator *username* și parola *pwd*;
- Sistemul de autentificare calculează $H(pwd)$ și se verifică dacă valoarea obținută este stocată în fișierul de parole pentru utilizatorul indicat prin *username*;
- Dacă da, atunci autentificarea se realizează cu succes; în caz contrar, autentificarea eșuază;
- Funcțiile hash sunt funcții **one-way**: cunoscând $H(pwd)$ nu se poate determina *pwd*;
- Această metodă de stocare a parolelor introduce deci avantajul că nu oferă adversarului acces direct la parole, chiar dacă acesta deține fișierul de parole.

Atac de tip dicționar

- Adversarul poate însă să verifice, pe rând, toate parolele cu probabilitate mare de apariție;

Atac de tip dicționar

- Adversarul poate însă să verifice, pe rând, toate parolele cu probabilitate mare de apariție;
- Acestea sunt de obicei cuvinte cu sens sau parole uzuale;

Atac de tip dicționar

- Adversarul poate însă să verifice, pe rând, toate parolele cu probabilitate mare de apariție;
- Acestea sunt de obicei cuvinte cu sens sau parole uzuale;
- Se consideră că formează termenii unui dicționar, de unde și numele atacului: **atac de tip dicționar**;

Atac de tip dicționar

- Adversarul poate însă să verifice, pe rând, toate parolele cu probabilitate mare de apariție;
- Acestea sunt de obicei cuvinte cu sens sau parole uzuale;
- Se consideră că formează termenii unui dicționar, de unde și numele atacului: **atac de tip dicționar**;
- Pentru a minimiza șansele unor astfel de atacuri:

Atac de tip dicționar

- Adversarul poate însă să verifice, pe rând, toate parolele cu probabilitate mare de apariție;
- Acestea sunt de obicei cuvinte cu sens sau parole uzuale;
- Se consideră că formează termenii unui dicționar, de unde și numele atacului: **atac de tip dicționar**;
- Pentru a minimiza șansele unor astfel de atacuri:
  - se blochează procesul de autentificare după un anumit număr de încercări nereușite;

Atac de tip dicționar

- Adversarul poate însă să verifice, pe rând, toate parolele cu probabilitate mare de apariție;
- Acestea sunt de obicei cuvinte cu sens sau parole uzuale;
- Se consideră că formează termenii unui dicționar, de unde și numele atacului: **atac de tip dicționar**;
- Pentru a minimiza șansele unor astfel de atacuri:
  - se blochează procesul de autentificare după un anumit număr de încercări nereușite;
  - se obligă utilizatorul să folosească o parolă care satisface anumite criterii : o lungime minimă, utilizarea a cel puțin 3 tipuri de simboluri (litere mici, litere mari, cifre, caractere speciale);

Atac de tip dicționar

- Adversarul poate însă să verifice, pe rând, toate parolele cu probabilitate mare de apariție;
- Acestea sunt de obicei cuvinte cu sens sau parole uzuale;
- Se consideră că formează termenii unui dicționar, de unde și numele atacului: **atac de tip dicționar**;
- Pentru a minimiza șansele unor astfel de atacuri:
  - se blochează procesul de autentificare după un anumit număr de încercări nereușite;
  - se obligă utilizatorul să folosească o parolă care satisface anumite criterii : o lungime minimă, utilizarea a cel puțin 3 tipuri de simboluri (litere mici, litere mari, cifre, caractere speciale);
- În caz de succes, adversarul determină parola unui singur utilizator;

Atac folosind tabele hash precalculate

- Pentru a determina parolele mai multor utilizatori simultan, un adversar poate **precalcula** valorile hash ale parolelor din dicționar;

Atac folosind tabele hash precalculate

- Pentru a determina parolele mai multor utilizatori simultan, un adversar poate **precalcula** valorile hash ale parolelor din dicționar;
- Dacă adversarul capătă acces la fișierul de parole, atunci verifică valorile care se regăsesc în lista precalculată;

Atac folosind tabele hash precalculate

- Pentru a determina parolele mai multor utilizatori simultan, un adversar poate **precalcula** valorile hash ale parolelor din dicționar;
- Dacă adversarul capătă acces la fișierul de parole, atunci verifică valorile care se regăsesc în lista precalculată;
- Toate conturile utilizatorilor pentru care se potrivesc valorile sunt compromise;

Atac folosind tabele hash precalculate

- Pentru a determina parolele mai multor utilizatori simultan, un adversar poate **precalcula** valorile hash ale parolelor din dicționar;
- Dacă adversarul capătă acces la fișierul de parole, atunci verifică valorile care se regăsesc în lista precalculată;
- Toate conturile utilizatorilor pentru care se potrivesc valorile sunt compromise;
- Atacul necesită capacitate de stocare mare: trebuie stocate toate perechile $(pwd, H(pwd))$ unde pwd este o parolă din dicționar;

Rainbow tables

- **Rainbow tables** introduc un compromis între capacitatea de stocare și timp;

Rainbow tables

- **Rainbow tables** introduc un compromis între capacitatea de stocare și timp;
- Se compun lanțuri de lungime t:
  $pwd_1 \rightarrow^H H(pwd_1) \rightarrow^f pwd_2 \rightarrow^H H(pwd_2) \rightarrow^f \cdots \rightarrow^f$
  $pwd_{t-1} \rightarrow^H H(pwd_{t-1}) \rightarrow^f pwd_t \rightarrow^H H(pwd_t)$

Rainbow tables

- **Rainbow tables** introduc un compromis între capacitatea de stocare și timp;
- Se compun lanțuri de lungime t:
  $pwd_1 \rightarrow^H H(pwd_1) \rightarrow^f pwd_2 \rightarrow^H H(pwd_2) \rightarrow^f \cdots \rightarrow^f$
  $pwd_{t-1} \rightarrow^H H(pwd_{t-1}) \rightarrow^f pwd_t \rightarrow^H H(pwd_t)$
- f este o funcție de mapare a valorilor hash în parole;

Rainbow tables

- **Rainbow tables** introduc un compromis între capacitatea de stocare și timp;
- Se compun lanțuri de lungime t:
  $pwd_1 \rightarrow^H H(pwd_1) \rightarrow^f pwd_2 \rightarrow^H H(pwd_2) \rightarrow^f \cdots \rightarrow^f$
  $pwd_{t-1} \rightarrow^H H(pwd_{t-1}) \rightarrow^f pwd_t \rightarrow^H H(pwd_t)$
- f este o funcție de mapare a valorilor hash în parole;
- Atenție! Funcția f nu este inversa funcției H (s-ar pierde proprietatea de *unidirecționalitate* a funcției hash)

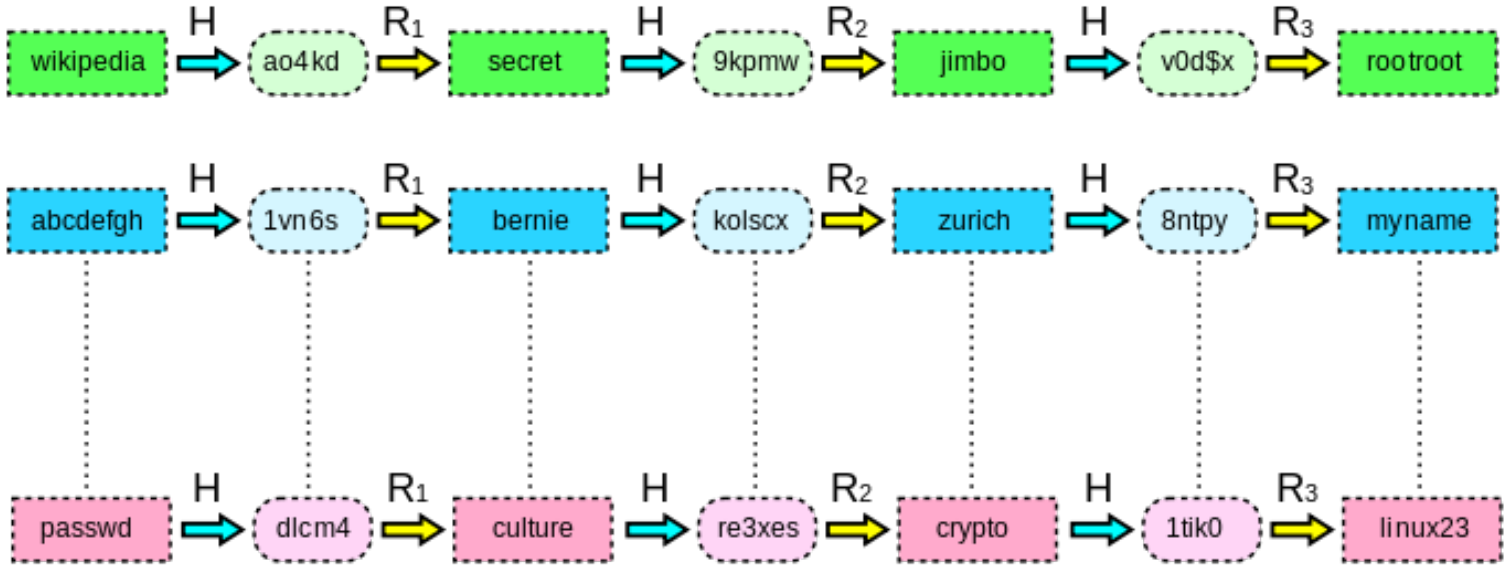
Rainbow tables

- **Rainbow tables** introduc un compromis între capacitatea de stocare și timp;
- Se compun lanțuri de lungime t:
  $pwd_1 \rightarrow^H H(pwd_1) \rightarrow^f pwd_2 \rightarrow^H H(pwd_2) \rightarrow^f \cdots \rightarrow^f$
  $pwd_{t-1} \rightarrow^H H(pwd_{t-1}) \rightarrow^f pwd_t \rightarrow^H H(pwd_t)$
- f este o funcție de mapare a valorilor hash în parole;
- Atenție! Funcția f nu este inversa funcției H (s-ar pierde proprietatea de *unidirecționalitate* a funcției hash)
- Se memorează doar capetele lanțurilor, valorile intermediare se generează la nevoie;

Rainbow tables

- **Rainbow tables** introduc un compromis între capacitatea de stocare și timp;
- Se compun lanțuri de lungime t:
  $pwd_1 \rightarrow^H H(pwd_1) \rightarrow^f pwd_2 \rightarrow^H H(pwd_2) \rightarrow^f \cdots \rightarrow^f$
  $pwd_{t-1} \rightarrow^H H(pwd_{t-1}) \rightarrow^f pwd_t \rightarrow^H H(pwd_t)$
- f este o funcție de mapare a valorilor hash în parole;
- Atenție! Funcția f nu este inversa funcției H (s-ar pierde proprietatea de *unidirecționalitate* a funcției hash)
- Se memorează doar capetele lanțurilor, valorile intermediare se generează la nevoie;
- Dacă t este suficient de mare, atunci capacitatea de stocare scade semnificativ;

Rainbow tables



[Wikipedia]

Rainbow tables

- Algoritmul de determinare a unei parole:

Rainbow tables

- Algoritmul de determinare a unei parole:

  1. Se caută valoarea hash a parolei în lista de valori hash a tabelei stocate;

Rainbow tables

- Algoritmul de determinare a unei parole:

  1. Se caută valoarea hash a parolei în lista de valori hash a tabelei stocate;
     1.1 Dacă se găsește, atunci lanțul căutat este acesta și se trece la pasul 2;

Rainbow tables

- Algoritmul de determinare a unei parole:

  1. Se caută valoarea hash a parolei în lista de valori hash a tabelei stocate;
     - 1.1 Dacă se găsește, atunci lanțul căutat este acesta și se trece la pasul 2;
     - 1.2 Dacă nu se găsește, se reduce valoarea hash prin funcția de reducere f într-o parolă căreia i se aplică funcția H și se reia cautarea de la pasul 1;

Rainbow tables

- Algoritmul de determinare a unei parole:

  1. Se caută valoarea hash a parolei în lista de valori hash a tabelei stocate;
     - 1.1 Dacă se găsește, atunci lanțul căutat este acesta și se trece la pasul 2;
     - 1.2 Dacă nu se găsește, se reduce valoarea hash prin funcția de reducere f într-o parolă căreia i se aplică funcția H și se reia cautarea de la pasul 1;
  2. Se generează întreg lanțul plecând de la valoarea inițială stocată. Parola corespunzătoare este cea situată în lanț înainte de valoarea hash căutată.

Salting

- Pentru a evita atacurile precedente, se utilizează tehnica de salting;

Salting

- Pentru a evita atacurile precedente, se utilizează tehnica de salting;
- La înregistrare, se stocheaza pentru fiecare utilizator:
$$(username, salt, H(pwd||salt))$$

Salting

- Pentru a evita atacurile precedente, se utilizează tehnica de salting;
- La înregistrare, se stocheaza pentru fiecare utilizator:
$$(username, salt, H(pwd||salt))$$
- *salt* este o secvență aleatoare de n biți, distinctă pentru fiecare utilizator;

Salting

- Pentru a evita atacurile precedente, se utilizează tehnica de salting;
- La înregistrare, se stocheaza pentru fiecare utilizator:
$$(username, salt, H(pwd||salt))$$
- *salt* este o secvență aleatoare de n biți, distinctă pentru fiecare utilizator;
- Adversarul nu poate precalcula valorile hash înainte de a obține acces la fișierul de parole...

Salting

- Pentru a evita atacurile precedente, se utilizează tehnica de salting;
- La înregistrare, se stocheaza pentru fiecare utilizator:
$$(username, salt, H(pwd||salt))$$
- *salt* este o secvență aleatoare de n biți, distinctă pentru fiecare utilizator;
- Adversarul nu poate precalcula valorile hash înainte de a obține acces la fișierul de parole...
- ... decât dacă folosește 2^n valori posibile *salt* pentru fiecare parolă;

Salting

- Pentru a evita atacurile precedente, se utilizează tehnica de salting;
- La înregistrare, se stocheaza pentru fiecare utilizator:
$$(username, salt, H(pwd||salt))$$
- *salt* este o secvență aleatoare de n biți, distinctă pentru fiecare utilizator;
- Adversarul nu poate precalcula valorile hash înainte de a obține acces la fișierul de parole...
- ... decât dacă folosește 2^n valori posibile *salt* pentru fiecare parolă;
- Atacurile devin deci impracticabile pentru n suficient de mare.

Salting

- În plus, în practică se folosesc funcții hash lente;

Salting

- În plus, în practică se folosesc funcții hash lente;
- Astfel verificarea unui număr mare de parole devine impracticabilă în timp real;

Salting

- În plus, în practică se folosesc funcții hash lente;
- Astfel verificarea unui număr mare de parole devine impracticabilă în timp real;
- Un alt avantaj introdus de tehnica de salting este că deși 2 utilizatori folosesc aceeași parolă, valorile stocate sunt diferite:

$$(Alice, 1652674, H(parolatest||1652674))$$
$$(Bob, 3154083, H(parolatest||3154083))$$

Salting

- În plus, în practică se folosesc funcții hash lente;
- Astfel verificarea unui număr mare de parole devine impracticabilă în timp real;
- Un alt avantaj introdus de tehnica de salting este că deși 2 utilizatori folosesc aceeași parolă, valorile stocate sunt diferite:

$$(Alice, 1652674, H(parolatest||1652674))$$
$$(Bob, 3154083, H(parolatest||3154083))$$

- Prin simpla citire a fișierului de parole, adversarul nu își poate da seama că 2 utilizatori folosesc aceeași parolă.

Parole de unică folosință

- Cazul extrem de schimbare periodică a parolei îl reprezintă parolele de unică folosință;

Parole de unică folosință

- Cazul extrem de schimbare periodică a parolei îl reprezintă parolele de unică folosință;
- Utilizatorul folosește o listă de parole, la fiecare logare utilizând următoarea parolă din listă;

Parole de unică folosință

- Cazul extrem de schimbare periodică a parolei îl reprezintă parolele de unică folosință;
- Utilizatorul folosește o listă de parole, la fiecare logare utilizând următoarea parolă din listă;
- Această lisă se calculează pornind de la o valoare x folosind o funcție hash H;

Parole de unică folosință

- Cazul extrem de schimbare periodică a parolei îl reprezintă parolele de unică folosință;
- Utilizatorul folosește o listă de parole, la fiecare logare utilizând următoarea parolă din listă;
- Această lisă se calculează pornind de la o valoare x folosind o funcție hash H;
- Să considerăm o listă de 3 parole de unică folosință:

$$P_0 = H(H(H(H(x))))$$

$$P_1 = H(H(H(x)))$$

$$P_2 = H(H(x))$$

$$P_3 = H(x)$$

Parole de unică folosință

- La înregistrare, în fișierul de parole se păstrează tripletul:
$$(username, 1, P_0 = H(H(H(H(x)))))$$

Parole de unică folosință

- La înregistrare, în fișierul de parole se păstrează tripletul:
$$(username, 1, P_0 = H(H(H(H(x)))))$$
- Utilizarorul introduce o parolă P_1 și autentificarea se realizează cu succes dacă $H(P_1) = P_0$;

Parole de unică folosință

- La înregistrare, în fișierul de parole se păstrează tripletul:
$$(username, 1, P_0 = H(H(H(H(x)))))$$
- Utilizarorul introduce o parolă P_1 și autentificarea se realizează cu succes dacă $H(P_1) = P_0$;
- Se actualizează fișierul de parole cu noua parolă introdusă:
$$(username, 2, P_1 = H(H(H(x))))$$

Parole de unică folosință

- La înregistrare, în fișierul de parole se păstrează tripletul:
$$(username, 1, P_0 = H(H(H(H(x)))))$$
- Utilizarorul introduce o parolă P_1 și autentificarea se realizează cu succes dacă $H(P_1) = P_0$;
- Se actualizează fișierul de parole cu noua parolă introdusă:
$$(username, 2, P_1 = H(H(H(x))))$$
- Procesul continuă până se ajunge la $H(x)$;

Parole de unică folosință

- La înregistrare, în fișierul de parole se păstrează tripletul:
$$(username, 1, P_0 = H(H(H(H(x)))))$$
- Utilizarorul introduce o parolă P_1 și autentificarea se realizează cu succes dacă $H(P_1) = P_0$;
- Se actualizează fișierul de parole cu noua parolă introdusă:
$$(username, 2, P_1 = H(H(H(x))))$$
- Procesul continuă până se ajunge la $H(x)$;
- Fiind cunoscută o parolă din secvență, se poate calcula imediat parola anterioară, dar NU se poate calcula parola următoare.

Important de reținut!

- Nu păstrați parolele în clar!
- Utilizați mecanisme de tip salting!