Criptografie și Securitate

- Prelegerea 17 -
Prezumpții criptografice dificile

Adela Georgescu, Ruxandra F. Olimid

Facultatea de Matematică și Informatică
Universitatea din București

Cuprins

Prezumpții criptografice dificile

- Criptografia modernă se bazează pe prezumpția că *anumite* probleme nu pot fi rezolvate în timp polinomial;

Prezumpții criptografice dificile

- Criptografia modernă se bazează pe prezumpția că *anumite* probleme nu pot fi rezolvate în timp polinomial;
- Până acum am văzut că schemele de criptare și de autentificare se bazează pe prezumpția existenței permutărilor pseudoaleatoare;

Prezumpții criptografice dificile

- Criptografia modernă se bazează pe prezumpția că *anumite* probleme nu pot fi rezolvate în timp polinomial;
- Până acum am văzut că schemele de criptare și de autentificare se bazează pe prezumpția existenței permutărilor pseudoaleatoare;
- Dar această prezumpție e nenaturală și foarte puternică;

Prezumpții criptografice dificile

- Criptografia modernă se bazează pe prezumpția că *anumite* probleme nu pot fi rezolvate în timp polinomial;
- Până acum am văzut că schemele de criptare și de autentificare se bazează pe prezumpția existenței permutărilor pseudoaleatoare;
- Dar această prezumpție e nenaturală și foarte puternică;
- În practică, PRF si PRP pot fi instanțiate cu cifruri bloc;

Prezumpții criptografice dificile

- Criptografia modernă se bazează pe prezumpția că *anumite* probleme nu pot fi rezolvate în timp polinomial;
- Până acum am văzut că schemele de criptare și de autentificare se bazează pe prezumpția existenței permutărilor pseudoaleatoare;
- Dar această prezumpție e nenaturală și foarte puternică;
- În practică, PRF si PRP pot fi instanțiate cu cifruri bloc;
- Însă metode pentru a demonstra pseudoaleatorismul construcțiilor practice relativ la alte prezumpții ”mai rezonabile” nu se cunosc;

Prezumpții criptografice dificile

- Criptografia modernă se bazează pe prezumpția că *anumite* probleme nu pot fi rezolvate în timp polinomial;
- Până acum am văzut că schemele de criptare și de autentificare se bazează pe prezumpția existenței permutărilor pseudoaleatoare;
- Dar această prezumpție e nenaturală și foarte puternică;
- În practică, PRF si PRP pot fi instanțiate cu cifruri bloc;
- Însă metode pentru a demonstra pseudoaleatorismul construcțiilor practice relativ la alte prezumpții "mai rezonabile" nu se cunosc;
- Dar e posibil a demonstra existența permutărilor pseudoaleatoare pe baza unei prezumpții mult mai slabe, cea a existenței funcțiilor one-way;

Prezumpții criptografice dificile

- În continuare vom introduce câteva probleme considerate "dificile" și vom prezenta funcții conjecturate ca fiind one-way bazate pe aceste probleme;

Prezumpții criptografice dificile

- În continuare vom introduce câteva probleme considerate "dificile" și vom prezenta funcții conjecturate ca fiind one-way bazate pe aceste probleme;
- Tot materialul ce urmează se bazează pe noțiuni de teoria numerelor;

Prezumpții criptografice dificile

- În continuare vom introduce câteva probleme considerate "dificile" și vom prezenta funcții conjecturate ca fiind one-way bazate pe aceste probleme;
- Tot materialul ce urmează se bazează pe noțiuni de teoria numerelor;
- La criptografia *simetrică* (cu cheie secretă) am văzut primitve criptografice (i.e. funcții hash, PRG, PRF, PRP) care pot fi construite eficient fără a implica teoria numerelor;

Prezumpții criptografice dificile

- În continuare vom introduce câteva probleme considerate "dificile" și vom prezenta funcții conjecturate ca fiind one-way bazate pe aceste probleme;
- Tot materialul ce urmează se bazează pe noțiuni de teoria numerelor;
- La criptografia *simetrică* (cu cheie secretă) am văzut primitve criptografice (i.e. funcții hash, PRG, PRF, PRP) care pot fi construite eficient fără a implica teoria numerelor;
- La criptografia *asimetrică* (cu cheie publică) construcțiile cunoscute se bazează pe probleme matematice dificile din teoria numerelor;

Primalitate și factorizare

- O primă problemă conjecturată ca fiind dificilă este problema factorizării numerelor întregi sau mai simplu problema factorizării;

Primalitate și factorizare

- O primă problemă conjecturată ca fiind dificilă este problema factorizării numerelor întregi sau mai simplu problema factorizării;
- Fiind dat un număr compus N, problema cere să se găsească două numere prime x_1 și x_2 pe n biți așa încât $N = x_1 \cdot x_2$;

Primalitate și factorizare

- O primă problemă conjecturată ca fiind dificilă este problema factorizării numerelor întregi sau mai simplu problema factorizării;
- Fiind dat un număr compus N, problema cere să se găsească două numere prime x_1 și x_2 pe n biți așa încât $N = x_1 \cdot x_2$;
- Cele mai greu de factorizat sunt numerele cele care au factori primi foarte mari.

Primalitate și factorizare

"The problem of distinguishing prime numbers from composite numbers and of resolving the latter into their prime factors is known to be one of the most important and useful in arithmetic. It has engaged the industry and wisdom of ancient and modern geometers to such an extent that it would be superfluous to discuss the problem at length... The dignity of the science itself seems to require that every possible means be explored for the solution of a problem so elegant and so celebrated."

(C.F.Gauss 1777 – 1855)

Generarea numerelor prime

- Pentru a putea folosi problema în criptografie, trebuie să generăm numere prime aleatoare *în mod eficient*;

Generarea numerelor prime

- Pentru a putea folosi problema în criptografie, trebuie să generăm numere prime aleatoare *în mod eficient*;
- Putem genera un numar prim aleator pe n biți prin alegerea repetată de numere aleatoare pe n biți până când găsim unul prim;

Generarea numerelor prime

- Pentru a putea folosi problema în criptografie, trebuie să generăm numere prime aleatoare *în mod eficient*;
- Putem genera un numar prim aleator pe n biți prin alegerea repetată de numere aleatoare pe n biți până când găsim unul prim;
- Pentru eficiență, ne interesează două aspecte:

Generarea numerelor prime

- Pentru a putea folosi problema în criptografie, trebuie să generăm numere prime aleatoare *în mod eficient*;
- Putem genera un numar prim aleator pe n biți prin alegerea repetată de numere aleatoare pe n biți până când găsim unul prim;
- Pentru eficiență, ne interesează două aspecte:
  1. probabilitatea ca un număr aleator de n biți să fie prim;

Generarea numerelor prime

- Pentru a putea folosi problema în criptografie, trebuie să generăm numere prime aleatoare *în mod eficient*;
- Putem genera un numar prim aleator pe n biți prin alegerea repetată de numere aleatoare pe n biți până când găsim unul prim;
- Pentru eficiență, ne interesează două aspecte:
  1. probabilitatea ca un număr aleator de n biți să fie prim;
  2. cum testăm eficient că un număr dat p este prim.

Generarea numerelor prime

- Pentru distribuția numerelor prime, se cunoaște următorul rezultat matematic:

Teoremă

Există o constantă c așa încât, pentru orice $n > 1$ numărul de numere prime pe n biți este cel puțin $c \cdot 2^{n-1}/n$

Generarea numerelor prime

- Pentru distribuția numerelor prime, se cunoaște următorul rezultat matematic:

Teoremă

Există o constantă c așa încât, pentru orice $n > 1$ numărul de numere prime pe n biți este cel puțin $c \cdot 2^{n-1}/n$

- Rezultă imediat că probabilitatea ca un numar ales aleator pe n biți să fie prim este cel putin c/n;

Generarea numerelor prime

- Pentru distribuția numerelor prime, se cunoaște următorul rezultat matematic:

Teoremă

Există o constantă c așa încât, pentru orice $n > 1$ numărul de numere prime pe n biți este cel puțin $c \cdot 2^{n-1}/n$

- Rezultă imediat că probabilitatea ca un numar ales aleator pe n biți să fie prim este cel putin c/n;
- Iar dacă testăm $t = n^2/c$ numere, probabilitatea ca un număr prim să nu fie ales este $(1 - c/n)^t$, adică cel mult e^{-n}, deci neglijabilă.

Testarea primalității

- Cei mai eficienți algoritmi sunt probabiliști:

Testarea primalității

- Cei mai eficienți algoritmi sunt probabiliști:
  - Dacă numărul p dat este prim atunci algoritmul întotdeauna întoarce rezultatul *prim*;

Testarea primalității

- Cei mai eficienți algoritmi sunt probabiliști:
  - Dacă numărul p dat este prim atunci algoritmul întotdeauna întoarce rezultatul *prim*;
  - Dacă p este compus, atunci cu probabilitate mare algoritmul va întoarce *compus*;

Testarea primalității

- Cei mai eficienți algoritmi sunt probabiliști:
  - Dacă numărul p dat este prim atunci algoritmul întotdeauna întoarce rezultatul *prim*;
  - Dacă p este compus, atunci cu probabilitate mare algoritmul va întoarce *compus*;
  - **Concluzie:** dacă outputul este *compus*, atunci p sigur este compus, dacă outputul este *prim*, atunci cu probabilitate mare p este prim dar este posibil și să se fi produs o eroare;

Testarea primalității

- Cei mai eficienți algoritmi sunt probabiliști:
  - Dacă numărul p dat este prim atunci algoritmul întotdeauna întoarce rezultatul *prim*;
  - Dacă p este compus, atunci cu probabilitate mare algoritmul va întoarce *compus*;
  - **Concluzie:** dacă outputul este *compus*, atunci p sigur este compus, dacă outputul este *prim*, atunci cu probabilitate mare p este prim dar este posibil și să se fi produs o eroare;
- Un algoritm determinist polinomial a fost propus în 2002, dar este mai lent decât algoritmii probabiliști;

Testarea primalității

- Cei mai eficienți algoritmi sunt probabiliști:
  - Dacă numărul p dat este prim atunci algoritmul întotdeauna întoarce rezultatul *prim*;
  - Dacă p este compus, atunci cu probabilitate mare algoritmul va întoarce *compus*;
  - **Concluzie:** dacă outputul este *compus*, atunci p sigur este compus, dacă outputul este *prim*, atunci cu probabilitate mare p este prim dar este posibil și să se fi produs o eroare;
- Un algoritm determinist polinomial a fost propus în 2002, dar este mai lent decât algoritmii probabiliști;
- Prezentăm un algoritm probabilist foarte răspândit: Miller-Rabin.

Testarea primalității

Algorithm 1 Algoritmul Miller-Rabin

Input: N , t

Output: o decizie dacă N este compus sau nu

if N este par sau $N = {N_1}^2$ **then**

    return "compus"

end if

compute $r \geq 1$ și u impar a.î. $N - 1 = 2^r u$

for $j = 1$ to t **do**

    $a \leftarrow \{1, ..., N-1\}$

    if $(gcd(a, N) \neq 1)$ **or** $(a^u \neq \pm 1 \bmod N$ **and** $a^{2^i u} \neq -1 \bmod N$, pentru orice $i \in \{1, ..., r-1\})$ **then**

        return "compus"

    end if

end for

return "prim"

Algoritmul Miller-Rabin

- Acceptă la intrare un numar N și un parametru t care determină *probabilitatea de eroare*;

Algoritmul Miller-Rabin

- Acceptă la intrare un numar N și un parametru t care determină *probabilitatea de eroare*;
- Rulează în timp **polinomial** în $|N|$ și t și satisface:

Algoritmul Miller-Rabin

- Acceptă la intrare un numar N și un parametru t care determină *probabilitatea de eroare*;
- Rulează în timp **polinomial** în $|N|$ și t și satisface:

Teoremă

Dacă N este prim, atunci testul Miller-Rabin întoarce mereu "prim". Dacă N este compus, algoritmul întoarce "prim" cu probabilitate cel mult 2^{-t} *(i.e. întoarce rezultatul corect "compus" cu probabilitate* $1 - 2^{-t}$*)*

Algoritmul Miller-Rabin

- Ne întoarcem la problema generării *eficiente* a numerelor prime;

Algoritmul Miller-Rabin

- Ne întoarcem la problema generării *eficiente* a numerelor prime;
- Folosim algoritmul Miller-Rabin pentru a descrie un algoritm **polinomial** de generare a numerelor prime;

Algoritmul Miller-Rabin

- Ne întoarcem la problema generării *eficiente* a numerelor prime;
- Folosim algoritmul Miller-Rabin pentru a descrie un algoritm **polinomial** de generare a numerelor prime;
- Pentru o intrare n, întoarce un număr aleator pe n biți care este prim cu excepția unei probabilități neglijabile în n.

Algoritmul de generare a numerelor prime

Algorithm 2 Algoritmul de generare a numerelor prime

Input: n

Output: Un număr aleator prim pe n biți

```
for i = 1 to n^2/c do
    p' ← {0,1}^(n−1)
    p = 1||p'
    execută testul Miller-Rabin pentru p cu parametrul n
    if output = "prim" then
        return p
    end if
end for
return eșec
```

Algoritmi de factorizare

- Deocamdată nu se cunosc *algoritmi polinomiali* pentru problema factorizării;

Algoritmi de factorizare

- Deocamdată nu se cunosc *algoritmi polinomiali* pentru problema factorizării;
- Dar există algoritmi mult mai buni decât forța brută;

Algoritmi de factorizare

- Deocamdată nu se cunosc *algoritmi polinomiali* pentru problema factorizării;
- Dar există algoritmi mult mai buni decât forța brută;
- Prezentăm în continuare câțiva algoritmi de factorizare.

Algoritmi de factorizare

- Reamintim: Fiind dat un număr compus N, **problema factorizării** cere să se găsească 2 numere prime p și q a.î. $N = pq$;

Algoritmi de factorizare

- Reamintim: Fiind dat un număr compus N, **problema factorizării** cere să se găsească 2 numere prime p și q a.î. $N = pq$;
- Considerăm $|p| = |q| = n$ și deci $n = O(\log N)$;

Algoritmi de factorizare

- Reamintim: Fiind dat un număr compus N, **problema factorizării** cere să se găsească 2 numere prime p și q a.î. $N = pq$;
- Considerăm $|p| = |q| = n$ și deci $n = O(\log N)$;
- Metoda cea mai simplă este împărțirea numărului N prin toate numerele p impare din intervalul $p = 3, ..., \left\lfloor \sqrt{N} \right\rfloor$.

Algoritmi de factorizare

- Reamintim: Fiind dat un număr compus N, **problema factorizării** cere să se găsească 2 numere prime p și q a.î. $N = pq$;
- Considerăm $|p| = |q| = n$ și deci $n = O(\log N)$;
- Metoda cea mai simplă este împărțirea numărului N prin toate numerele p impare din intervalul $p = 3, ..., \left\lfloor \sqrt{N} \right\rfloor$.
- Complexitatea timp este $O(\sqrt{N} \cdot (\log N)^c)$ unde c este o constantă;

Algoritmi de factorizare

- Reamintim: Fiind dat un număr compus N, **problema factorizării** cere să se găsească 2 numere prime p și q a.î. $N = pq$;
- Considerăm $|p| = |q| = n$ și deci $n = O(\log N)$;
- Metoda cea mai simplă este împărțirea numărului N prin toate numerele p impare din intervalul $p = 3, ..., \left\lfloor \sqrt{N} \right\rfloor$.
- Complexitatea timp este $O(\sqrt{N} \cdot (\log N)^c)$ unde c este o constantă;
- Pentru $N < 10^{12}$ metoda este destul de eficientă.

Algoritmi de factorizare

- Există însă algoritmi mai sofisticați, cu timp de execuție mai bun, între care:

Algoritmi de factorizare

- Există însă algoritmi mai sofisticați, cu timp de execuție mai bun, între care:
  - Metoda **Pollard $\mathbf{p-1}$**: funcționează atunci când $p-1$ are factori primi "mici";

Algoritmi de factorizare

- Există însă algoritmi mai sofisticați, cu timp de execuție mai bun, între care:
  - Metoda **Pollard $\mathbf{p-1}$**: funcționează atunci când $p-1$ are factori primi "mici";
  - Metoda **Pollard rho**: timpul de execuție este $O(N^{1/4} \cdot (\log N)^c)$;

Algoritmi de factorizare

- Există însă algoritmi mai sofisticați, cu timp de execuție mai bun, între care:
  - Metoda **Pollard $\mathbf{p-1}$**: funcționează atunci când $p-1$ are factori primi "mici";
  - Metoda **Pollard rho**: timpul de execuție este $O(N^{1/4} \cdot (\log N)^c)$;
  - **Algoritmul sitei pătratice** - rulează în timp *sub-exponențial* în lungimea lui N.

Algoritmi de factorizare

- Există însă algoritmi mai sofisticați, cu timp de execuție mai bun, între care:
  - Metoda **Pollard $\mathbf{p-1}$**: funcționează atunci când $p-1$ are factori primi "mici";
  - Metoda **Pollard rho**: timpul de execuție este $O(N^{1/4} \cdot (\log N)^c)$;
  - **Algoritmul sitei pătratice** - rulează în timp *sub-exponențial* în lungimea lui N.
- Deocamdată, cel mai rapid algoritm de factorizare este o îmbunătățire a sitei pătratice care factorizează un număr N de lungime $O(n)$ în timp $2^{O(n^{1/3} \cdot (\log n)^{2/3})}$.

Algoritmul sitei pătratice

- Un element $y \in \mathbb{Z}_p^*$ este *rest pătratic modulo p* dacă $\exists x \in \mathbb{Z}_p^*$ a.î.

$$x^2 = y \bmod p$$

Algoritmul sitei pătratice

- Un element $y \in \mathbb{Z}_p^*$ este *rest pătratic modulo p* dacă $\exists x \in \mathbb{Z}_p^*$ a.î.

$$x^2 = y \bmod p$$

- x se numește *rădăcina pătratică* a lui y;

Algoritmul sitei pătratice

- Un element $y \in \mathbb{Z}_p^*$ este *rest pătratic modulo p* dacă $\exists x \in \mathbb{Z}_p^*$ a.î.

$$x^2 = y \bmod p$$

- x se numește *rădăcina pătratică* a lui y;
- Algoritmul se bazează pe două observații simple:

Algoritmul sitei pătratice

- Un element $y \in \mathbb{Z}_p^*$ este *rest pătratic modulo p* dacă $\exists x \in \mathbb{Z}_p^*$ a.î.

$$x^2 = y \bmod p$$

- x se numește *rădăcina pătratică* a lui y;
- Algoritmul se bazează pe două observații simple:
  1. Dacă $N = pq$ cu p, q prime distincte, atunci fiecare rest pătratic modulo N are exact 4 rădăcini pătratice diferite;

Algoritmul sitei pătratice

- Un element $y \in \mathbb{Z}_p^*$ este *rest pătratic modulo p* dacă $\exists x \in \mathbb{Z}_p^*$ a.î.
$$x^2 = y \bmod p$$
- x se numește *rădăcina pătratică* a lui y;
- Algoritmul se bazează pe două observații simple:
  1. Dacă $N = pq$ cu p, q prime distincte, atunci fiecare rest pătratic modulo N are exact 4 rădăcini pătratice diferite;
  2. Dacă se pot afla x, y cu $x^2 = y^2 \bmod N$ si $x \neq \pm y \bmod N$, atunci $gcd(x - y, N)$ este un factor prim al lui N calculabil în timp polinomial;

Algoritmul sitei pătratice

- Algoritmul încearcă să genereze o pereche de valori x, y a.î. $x^2 = y^2 \bmod N$, în speranța că $x \neq \pm y \bmod N$ cu probabilitate constantă; căutarea se desfășoară astfel:

Algoritmul sitei pătratice

- Algoritmul încearcă să genereze o pereche de valori x, y a.î. $x^2 = y^2 \bmod N$, în speranța că $x \neq \pm y \bmod N$ cu probabilitate constantă; căutarea se desfășoară astfel:
- Se fixează o bază $B = \{p_1, ..., p_k\}$ de numere prime mici;

Algoritmul sitei pătratice

- Algoritmul încearcă să genereze o pereche de valori x, y a.î. $x^2 = y^2 \bmod N$, în speranța că $x \neq \pm y \bmod N$ cu probabilitate constantă; căutarea se desfășoară astfel:
- Se fixează o bază $B = \{p_1, ..., p_k\}$ de numere prime mici;
- Se caută $l > k$ numere distincte $x_1, ..., x_l \in \mathbb{Z}_N^*$ pentru care $q_i = [x_i^2 \bmod N]$ este "mic" așa încât toți factorii primi ai lui q_i se găsesc în B;

Algoritmul sitei pătratice

- Algoritmul încearcă să genereze o pereche de valori x, y a.î. $x^2 = y^2 \bmod N$, în speranța că $x \neq \pm y \bmod N$ cu probabilitate constantă; căutarea se desfășoară astfel:
- Se fixează o bază $B = \{p_1, ..., p_k\}$ de numere prime mici;
- Se caută $l > k$ numere distincte $x_1, ..., x_l \in \mathbb{Z}_N^*$ pentru care $q_i = [x_i^2 \bmod N]$ este "mic" așa încât toți factorii primi ai lui q_i se găsesc în B;
- Vor rezulta relații de forma

$$x_j^2 = p_1^{e_{j,1}} \cdot p_2^{e_{j,2}} \cdot \cdots \cdot p_k^{e_{j,k}} \bmod N$$

unde $1 \leq j \leq k$.

Algoritmul sitei pătratice

- Pentru fiecare j se consideră vectorul:
$$e_j = (e_{j,1} \bmod 2, ..., e_{j,k} \bmod 2)$$

Algoritmul sitei pătratice

- Pentru fiecare j se consideră vectorul:
$$e_j = (e_{j,1} \bmod 2, ..., e_{j,k} \bmod 2)$$
- Dacă se poate determina o submulțime formată din astfel de vectori, a căror suma modulo 2 să fie (0,0,...,0), atunci pătratul produsului elementelor x_j corespunzătoare va avea în B toți divizorii reprezentați de un număr par de ori.

Algoritmul sitei pătratice

- Pentru fiecare j se consideră vectorul:
$$e_j = (e_{j,1} \bmod 2, ..., e_{j,k} \bmod 2)$$
- Dacă se poate determina o submulțime formată din astfel de vectori, a căror suma modulo 2 să fie (0,0,...,0), atunci pătratul produsului elementelor x_j corespunzătoare va avea în B toți divizorii reprezentați de un număr par de ori.
- Se poate arăta că algoritmul optimizat rulează în timp $2^{O(\sqrt{n \cdot logn})}$ pentru factorizarea unui număr N de lungime $O(n)$, deci **sub-exponențial** în lungimea lui N.

Exemplu

- Fie N $= 377753$. Se știe că $6647 = [620^2 \bmod N]$ și putem factoriza
$$6647 = 17^2 \cdot 23$$

Exemplu

- Fie N = 377753. Se știe că $6647 = [620^2 \bmod N]$ și putem factoriza
$$6647 = 17^2 \cdot 23$$
- Deci:
$$620^2 = 17^2 \cdot 23 \bmod N$$

Exemplu

- Fie N = 377753. Se știe că $6647 = [620^2 \bmod N]$ și putem factoriza
$$6647 = 17^2 \cdot 23$$
- Deci:
$$620^2 = 17^2 \cdot 23 \bmod N$$
- Similar:
$$621^2 = 2^4 \cdot 17 \cdot 29 \bmod N$$
$$645^2 = 2^7 \cdot 13 \cdot 23 \bmod N$$
$$655^2 = 2^3 \cdot 13 \cdot 27 \cdot 29 \bmod N$$

Exemplu

- Fie N $= 377753$. Se știe că $6647 = [620^2 \bmod N]$ și putem factoriza
$$6647 = 17^2 \cdot 23$$
- Deci:
$$620^2 = 17^2 \cdot 23 \bmod N$$
- Similar:
$$621^2 = 2^4 \cdot 17 \cdot 29 \bmod N$$
$$645^2 = 2^7 \cdot 13 \cdot 23 \bmod N$$
$$655^2 = 2^3 \cdot 13 \cdot 27 \cdot 29 \bmod N$$
- Baza de numere prime mici este:
$$B = \{2, 13, 17, 23, 29\}$$

Exemplu

- Obținem:

$$620^2 \cdot 621^2 \cdot 645^2 \cdot 655^2 = 2^{14} \cdot 13^2 \cdot 17^4 \cdot 23^2 \cdot 29^2 \bmod N$$

$$[620 \cdot 621 \cdot 645 \cdot 655 \bmod N]^2 = [2^7 \cdot 13 \cdot 17^2 \cdot 23 \cdot 29 \bmod N]^2 \bmod N$$

Exemplu

- Obținem:

$$620^2 \cdot 621^2 \cdot 645^2 \cdot 655^2 = 2^{14} \cdot 13^2 \cdot 17^4 \cdot 23^2 \cdot 29^2 \bmod N$$

$$[620 \cdot 621 \cdot 645 \cdot 655 \bmod N]^2 = [2^7 \cdot 13 \cdot 17^2 \cdot 23 \cdot 29 \bmod N]^2 \bmod N$$

- Toți exponenții sunt pari!

Exemplu

- Obținem:

$$620^2 \cdot 621^2 \cdot 645^2 \cdot 655^2 = 2^{14} \cdot 13^2 \cdot 17^4 \cdot 23^2 \cdot 29^2 \bmod N$$

$$[620 \cdot 621 \cdot 645 \cdot 655 \bmod N]^2 = [2^7 \cdot 13 \cdot 17^2 \cdot 23 \cdot 29 \bmod N]^2 \bmod N$$

- Toți exponenții sunt pari!
- După efectuarea calculelor:

$$\Rightarrow 127194^2 = 45335^2 \bmod N$$

Exemplu

- Obținem:

$$620^2 \cdot 621^2 \cdot 645^2 \cdot 655^2 = 2^{14} \cdot 13^2 \cdot 17^4 \cdot 23^2 \cdot 29^2 \bmod N$$

$$[620 \cdot 621 \cdot 645 \cdot 655 \bmod N]^2 = [2^7 \cdot 13 \cdot 17^2 \cdot 23 \cdot 29 \bmod N]^2 \bmod N$$

- Toți exponenții sunt pari!
- După efectuarea calculelor:

$$\Rightarrow 127194^2 = 45335^2 \bmod N$$

- Cum $127194 \neq 45335 \bmod N$, se calculează un factor prim al lui N:

$$gcd(127194 - 45335, 377753) = 751$$

RSA Challenge

- În 1991, Laboratoarele RSA lansează *RSA Challenge*;

RSA Challenge

- În 1991, Laboratoarele RSA lansează *RSA Challenge*;
- Aceasta presupune factorizarea unor numere N, unde $N = p \cdot q$, cu p, q 2 numere prime mari;

RSA Challenge

- În 1991, Laboratoarele RSA lansează *RSA Challenge*;
- Aceasta presupune factorizarea unor numere N, unde $N = p \cdot q$, cu p, q 2 numere prime mari;
- Au fost lansate mai multe provocări, câte 1 pentru fiecare dimensiune (în biți) a lui N:

RSA Challenge

- În 1991, Laboratoarele RSA lansează *RSA Challenge*;
- Aceasta presupune factorizarea unor numere N, unde $N = p \cdot q$, cu p, q 2 numere prime mari;
- Au fost lansate mai multe provocări, câte 1 pentru fiecare dimensiune (în biți) a lui N:
- Exemple includ: RSA-576, RSA-640, RSA-768, $\cdots$, RSA-1024, RSA-1536, RSA-2048;

RSA Challenge

- În 1991, Laboratoarele RSA lansează *RSA Challenge*;
- Aceasta presupune factorizarea unor numere N, unde $N = p \cdot q$, cu p, q 2 numere prime mari;
- Au fost lansate mai multe provocări, câte 1 pentru fiecare dimensiune (în biți) a lui N:
- Exemple includ: RSA-576, RSA-640, RSA-768, $\cdots$, RSA-1024, RSA-1536, RSA-2048;
- Provocarea s-a încheiat oficial în 2007;

RSA Challenge

- În 1991, Laboratoarele RSA lansează *RSA Challenge*;
- Aceasta presupune factorizarea unor numere N, unde $N = p \cdot q$, cu p, q 2 numere prime mari;
- Au fost lansate mai multe provocări, câte 1 pentru fiecare dimensiune (în biți) a lui N:
- Exemple includ: RSA-576, RSA-640, RSA-768, $\cdots$, RSA-1024, RSA-1536, RSA-2048;
- Provocarea s-a încheiat oficial în 2007;
- Multe provocări au fost sparte în cursul anilor (chiar și ulterior închiderii oficiale), însă există numere încă nefactorizate:

RSA Challenge

- **RSA-1024**

  135066410865995223349603216278805969938881475605667
  027524485143851526510604859533833940287150571909441
  798207282164471551373680419703964191743046496589274
  256239341020864383202110372958725762358509643110564
  073501508187510676594629205563685529475213500852879
  416377328533906109750544334999811150056977236
  890927563

  [http://www.emc.com/emc-plus/rsa-labs/historical/the-rsa-challenge-numbers.htm]

RSA Challenge

- **RSA-2048**

  25195908475657893494027183240048398571429282126204
  03202777713783604366202070759555626401852588078440
  69182906412495150821892985591491761845028084891200
  72844992687392807287776735971418347270261896375014
  97182469116507761337985909570009733045974880842840
  17974291006424586918171951187461215151726546322822
  16869987549182422433637259085141865462043576798423
  38718477444792073993423658482382428119816381501067
  48104516603773060562016196762561338441436038339044
  14952634432190114657544454178424020924616515723350
  77870774981712577246796292638635637328991215483143
  81678998850404453640235273819513786365643912120103
  97122822120720357

  [http://www.emc.com/emc-plus/rsa-labs/historical/the-rsa-challenge-numbers.htm]

Prezumpția logaritmului discret

- O altă prezumție dificilă este DLP (Discrete Logarithm Problem) (sau PLD (Problema Logaritmului Discret));

Prezumpția logaritmului discret

- O altă prezumție dificilă este DLP (Discrete Logarithm Problem) (sau PLD (Problema Logaritmului Discret));
- Considerăm $\mathbb{G}$ un grup ciclic de ordin q;

Prezumpția logaritmului discret

- O altă prezumție dificilă este DLP (Discrete Logarithm Problem) (sau PLD (Problema Logaritmului Discret));
- Considerăm $\mathbb{G}$ un grup ciclic de ordin q;
- Există un generator $g \in \mathbb{G}$ a.î. $\mathbb{G} = \{g^0, g^1, ..., g^{q-1}\}$;

Prezumpția logaritmului discret

- O altă prezumție dificilă este DLP (Discrete Logarithm Problem) (sau PLD (Problema Logaritmului Discret));
- Considerăm $\mathbb{G}$ un grup ciclic de ordin q;
- Există un generator $g \in \mathbb{G}$ a.î. $\mathbb{G} = \{g^0, g^1, ..., g^{q-1}\}$;
- Echivalent, pentru fiecare $h \in \mathbb{G}$ există un *unic* $x \in \mathbb{Z}_q$ a.î. $g^x = h$;

Prezumpția logaritmului discret

- O altă prezumție dificilă este DLP (Discrete Logarithm Problem) (sau PLD (Problema Logaritmului Discret));
- Considerăm $\mathbb{G}$ un grup ciclic de ordin q;
- Există un generator $g \in \mathbb{G}$ a.î. $\mathbb{G} = \{g^0, g^1, ..., g^{q-1}\}$;
- Echivalent, pentru fiecare $h \in \mathbb{G}$ există un *unic* $x \in \mathbb{Z}_q$ a.î. $g^x = h$;
- x se numește logaritmul discret al lui h în raport cu g și se notează

$$x = \log_g h$$

Experimentul logaritmului discret $DLog_{\mathcal{A}}(n)$

1. Generează $(\mathbb{G}, q, g)$ unde $\mathbb{G}$ este un grup ciclic de ordin q (cu $|q| = n$) iar g este generatorul lui $\mathbb{G}$.
2. Alege $h \leftarrow^{R} \mathbb{G}$. (se poate alege $x' \leftarrow^{R} \mathbb{Z}_q$ și apoi $h := g^{x'}$.)

Experimentul logaritmului discret $DLog_{\mathcal{A}}(n)$

1. Generează $(\mathbb{G}, q, g)$ unde $\mathbb{G}$ este un grup ciclic de ordin q (cu $|q| = n$) iar g este generatorul lui $\mathbb{G}$.
2. Alege $h \leftarrow^R \mathbb{G}$. (se poate alege $x' \leftarrow^R \mathbb{Z}_q$ și apoi $h := g^{x'}$.)
3. $\mathcal{A}$ primește $\mathbb{G}, q, g, h$ și întoarce $x \in \mathbb{Z}_q$;

Experimentul logaritmului discret $DLog_{\mathcal{A}}(n)$

1. Generează $(\mathbb{G}, q, g)$ unde $\mathbb{G}$ este un grup ciclic de ordin q (cu $|q| = n$) iar g este generatorul lui $\mathbb{G}$.
2. Alege $h \leftarrow^R \mathbb{G}$. (se poate alege $x' \leftarrow^R \mathbb{Z}_q$ și apoi $h := g^{x'}$.)
3. $\mathcal{A}$ primește $\mathbb{G}, q, g, h$ și întoarce $x \in \mathbb{Z}_q$;
4. Output-ul experimentului este 1 dacă $g^x = h$ și 0 altfel.

Experimentul logaritmului discret $DLog_{\mathcal{A}}(n)$

1. Generează $(\mathbb{G}, q, g)$ unde $\mathbb{G}$ este un grup ciclic de ordin q (cu $|q| = n$) iar g este generatorul lui $\mathbb{G}$.
2. Alege $h \leftarrow^R \mathbb{G}$. (se poate alege $x' \leftarrow^R \mathbb{Z}_q$ și apoi $h := g^{x'}$.)
3. $\mathcal{A}$ primește $\mathbb{G}, q, g, h$ și întoarce $x \in \mathbb{Z}_q$;
4. Output-ul experimentului este 1 dacă $g^x = h$ și 0 altfel.

Definiție

Spunem că problema logaritmului discret (DLP) este dificilă dacă pentru orice algoritm PPT $\mathcal{A}$ există o funcție neglijabilă negl așa încât

$$Pr[DLog_{\mathcal{A}}(n) = 1] \leq negl(n)$$

Grupuri ciclice de ordin prim

- Există câteva clase de grupuri ciclice pentru care DLP este considerată dificilă;

Grupuri ciclice de ordin prim

- Există câteva clase de grupuri ciclice pentru care DLP este considerată dificilă;
- Una dintre ele este clasa grupurilor ciclice de *ordin prim* (în aceste grupuri, problema este "cea mai dificilă");

Grupuri ciclice de ordin prim

- Există câteva clase de grupuri ciclice pentru care DLP este considerată dificilă;
- Una dintre ele este clasa grupurilor ciclice de *ordin prim* (în aceste grupuri, problema este "cea mai dificilă");
- DLP nu poate fi rezolvată în timp polinomial în grupurile care nu sunt de ordin prim, ci doar este *mai ușoară*;

Grupuri ciclice de ordin prim

- Există câteva clase de grupuri ciclice pentru care DLP este considerată dificilă;
- Una dintre ele este clasa grupurilor ciclice de *ordin prim* (în aceste grupuri, problema este "cea mai dificilă");
- DLP nu poate fi rezolvată în timp polinomial în grupurile care nu sunt de ordin prim, ci doar este *mai ușoară*;
- În aceste grupuri căutarea unui generator și verificarea că un număr dat este generator sunt triviale.

Lucrul în $\mathbb{Z}_p^*$

- DLP este considerată dificilă în grupuri ciclice de forma $\mathbb{Z}_p^*$ cu p prim;

Lucrul în $\mathbb{Z}_p^*$

- DLP este considerată dificilă în grupuri ciclice de forma $\mathbb{Z}_p^*$ cu p prim;
- Insă pentru $p > 3$ grupul $\mathbb{Z}_p^*$ NU are ordin prim;

Lucrul în $\mathbb{Z}_p^*$

- DLP este considerată dificilă în grupuri ciclice de forma $\mathbb{Z}_p^*$ cu p prim;
- Insă pentru $p > 3$ grupul $\mathbb{Z}_p^*$ NU are ordin prim;
- Aceasta problemă se rezolvă folosind un *subgrup* potrivit al lui $\mathbb{Z}_p^*$;

Lucrul în $\mathbb{Z}_p^*$

- DLP este considerată dificilă în grupuri ciclice de forma $\mathbb{Z}_p^*$ cu p prim;
- Insă pentru $p > 3$ grupul $\mathbb{Z}_p^*$ NU are ordin prim;
- Aceasta problemă se rezolvă folosind un *subgrup* potrivit al lui $\mathbb{Z}_p^*$;
- Reamintim: Un element $y \in \mathbb{Z}_p^*$ este *rest pătratic modulo p* dacă $\exists x \in \mathbb{Z}_p^*$ a.î. $x^2 = y \bmod p$;

Lucrul în $\mathbb{Z}_p^*$

- DLP este considerată dificilă în grupuri ciclice de forma $\mathbb{Z}_p^*$ cu p prim;
- Insă pentru $p > 3$ grupul $\mathbb{Z}_p^*$ NU are ordin prim;
- Aceasta problemă se rezolvă folosind un *subgrup* potrivit al lui $\mathbb{Z}_p^*$;
- Reamintim: Un element $y \in \mathbb{Z}_p^*$ este *rest pătratic modulo p* dacă $\exists x \in \mathbb{Z}_p^*$ a.î. $x^2 = y \bmod p$;
- Mulțimea resturilor pătratice modulo p formează un subgrup al lui $\mathbb{Z}_p^*$;

Lucrul în $\mathbb{Z}_p^*$

- DLP este considerată dificilă în grupuri ciclice de forma $\mathbb{Z}_p^*$ cu p prim;
- Insă pentru $p > 3$ grupul $\mathbb{Z}_p^*$ NU are ordin prim;
- Aceasta problemă se rezolvă folosind un *subgrup* potrivit al lui $\mathbb{Z}_p^*$;
- Reamintim: Un element $y \in \mathbb{Z}_p^*$ este *rest pătratic modulo p* dacă $\exists x \in \mathbb{Z}_p^*$ a.î. $x^2 = y \bmod p$;
- Mulțimea resturilor pătratice modulo p formează un subgrup al lui $\mathbb{Z}_p^*$;
- Dacă p este număr *prim tare* (i.e. $p = 2q + 1$ cu q prim), subgrupul resturilor pătratice modulo p are ordin q, deci este ciclic și toate elementele sunt generatori.

Important de reținut!

- Cel mai rapid algoritm de factorizare necesită timp sub-exponențial;
- Problema logaritmului discret este dificilă.