Criptografie și Securitate

- Prelegerea 22 -
Despre problema logaritmului discret

Adela Georgescu, Ruxandra F. Olimid

Facultatea de Matematică și Informatică
Universitatea din București

Cuprins

Algoritmi pentru calculul logaritmului discret

- Reamintim PLD:

Algoritmi pentru calculul logaritmului discret

- Reamintim PLD:
- Fie $\mathbb{G}$ un grup ciclic de ordin q (cu $|q| = n$) iar g este generatorul lui $\mathbb{G}$.

Algoritmi pentru calculul logaritmului discret

- Reamintim PLD:
- Fie $\mathbb{G}$ un grup ciclic de ordin q (cu $|q| = n$) iar g este generatorul lui $\mathbb{G}$.
- Pentru fiecare $h \in \mathbb{G}$ există un unic $x \in \mathbb{Z}_q$ a.î. $g^x = h$.

Algoritmi pentru calculul logaritmului discret

- Reamintim PLD:
- Fie $\mathbb{G}$ un grup ciclic de ordin q (cu $|q| = n$) iar g este generatorul lui $\mathbb{G}$.
- Pentru fiecare $h \in \mathbb{G}$ există un unic $x \in \mathbb{Z}_q$ a.î. $g^x = h$.
- PLD cere găsirea lui x știind $\mathbb{G}, q, g, h$; notăm $x = \log_g h$;

Algoritmi pentru calculul logaritmului discret

- Reamintim PLD:
- Fie $\mathbb{G}$ un grup ciclic de ordin q (cu $|q| = n$) iar g este generatorul lui $\mathbb{G}$.
- Pentru fiecare $h \in \mathbb{G}$ există un unic $x \in \mathbb{Z}_q$ a.î. $g^x = h$.
- PLD cere găsirea lui x știind $\mathbb{G}, q, g, h$; notăm $x = \log_g h$;
- Atenție! Atunci când $g^{x'} = h$ pentru un x' arbitrar (deci NU neapărat $x \in \mathbb{Z}_q$), notăm $log_g h = [x' \bmod q]$

Algoritmi pentru calculul logaritmului discret

- Problema PLD se poate rezolva, desigur, prin forță brută, calculând pe rând toate puterile x ale lui g până când se găsește una potrivită pentru care $g^x = h$;

Algoritmi pentru calculul logaritmului discret

- Problema PLD se poate rezolva, desigur, prin forță brută, calculând pe rând toate puterile x ale lui g până când se găsește una potrivită pentru care $g^x = h$;
- Complexitatea timp este $\mathcal{O}(q)$ iar complexitatea spațiu este $\mathcal{O}(1)$;

Algoritmi pentru calculul logaritmului discret

- Problema PLD se poate rezolva, desigur, prin forță brută, calculând pe rând toate puterile x ale lui g până când se găsește una potrivită pentru care $g^x = h$;
- Complexitatea timp este $\mathcal{O}(q)$ iar complexitatea spațiu este $\mathcal{O}(1)$;
- Dacă se precalculează toate valorile (x, g^x), căutarea se face în timp $\mathcal{O}(1)$ și spațiu $\mathcal{O}(q)$;

Algoritmi pentru calculul logaritmului discret

- Problema PLD se poate rezolva, desigur, prin forță brută, calculând pe rând toate puterile x ale lui g până când se găsește una potrivită pentru care $g^x = h$;
- Complexitatea timp este $\mathcal{O}(q)$ iar complexitatea spațiu este $\mathcal{O}(1)$;
- Dacă se precalculează toate valorile (x, g^x), căutarea se face în timp $\mathcal{O}(1)$ și spațiu $\mathcal{O}(q)$;
- Sunt de interes algoritmii care pot obține un timp mai bun la rulare decât forța brută, realizând un compromis spațiu-timp.

Algoritmi pentru calculul logaritmului discret

- Se cunosc mai mulți astfel de algoritmi împărțiți în două categorii:

Algoritmi pentru calculul logaritmului discret

- Se cunosc mai mulți astfel de algoritmi împărțiți în două categorii:
  - algoritmi *generici* care funcționează în grupuri arbitrare (i.e. orice grupuri ciclice);

Algoritmi pentru calculul logaritmului discret

- Se cunosc mai mulți astfel de algoritmi împărțiți în două categorii:
  - algoritmi *generici* care funcționează în grupuri arbitrare (i.e. orice grupuri ciclice);
  - algoritmi *non-generici* care lucrează în grupuri *specifice* - exploatează proprietăți speciale ale anumitor grupuri

Algoritmi pentru calculul logaritmului discret

- Se cunosc mai mulți astfel de algoritmi împărțiți în două categorii:
  - algoritmi *generici* care funcționează în grupuri arbitrare (i.e. orice grupuri ciclice);
  - algoritmi *non-generici* care lucrează în grupuri *specifice* - exploatează proprietăți speciale ale anumitor grupuri

- Dintre algoritmii generici enumerăm:

Algoritmi pentru calculul logaritmului discret

- Se cunosc mai mulți astfel de algoritmi împărțiți în două categorii:
  - algoritmi *generici* care funcționează în grupuri arbitrare (i.e. orice grupuri ciclice);
  - algoritmi *non-generici* care lucrează în grupuri *specifice* - exploatează proprietăți speciale ale anumitor grupuri

- Dintre algoritmii generici enumerăm:

- Metoda **Baby-step/giant-step**, datorată lui Shanks, calculează logaritmul discret într-un grup de ordin q în timp $\mathcal{O}(\sqrt{q} \cdot (\log q)^c)$;

Algoritmi pentru calculul logaritmului discret

- Se cunosc mai mulți astfel de algoritmi împărțiți în două categorii:
  - algoritmi *generici* care funcționează în grupuri arbitrare (i.e. orice grupuri ciclice);
  - algoritmi *non-generici* care lucrează în grupuri *specifice* - exploatează proprietăți speciale ale anumitor grupuri
- Dintre algoritmii generici enumerăm:
- Metoda **Baby-step/giant-step**, datorată lui Shanks, calculează logaritmul discret într-un grup de ordin q în timp $\mathcal{O}(\sqrt{q} \cdot (\log q)^c)$;
- Algoritmul **Pohlig-Hellman** poate fi folosit atunci când se cunoaște factorizarea ordinului q al grupului;

Algoritmi generici pentru calculul logaritmului discret

- Metoda Baby-Step/Giant-Step este optimă ca timp de rulare, însă există alți algoritmi mai eficienți d.p.d.v. al complexității spațiu;

Algoritmi generici pentru calculul logaritmului discret

- Metoda Baby-Step/Giant-Step este optimă ca timp de rulare, însă există alți algoritmi mai eficienți d.p.d.v. al complexității spațiu;
- În cazul algoritmului Pohlig-Hellman, timpul de rulare depinde factorii primi ai lui q;

Algoritmi generici pentru calculul logaritmului discret

- Metoda Baby-Step/Giant-Step este optimă ca timp de rulare, însă există alți algoritmi mai eficienți d.p.d.v. al complexității spațiu;
- În cazul algoritmului Pohlig-Hellman, timpul de rulare depinde factorii primi ai lui q;
- Pentru ca algoritmul să nu fie eficient, trebuie ca cel mai mare factor prim al lui q să fie de ordinul 2^{160}.

Algoritmi non-generici pentru calculul logaritmului discret

- Algoritmii non-generici sunt potențial mai puternici decât cei generici;

Algoritmi non-generici pentru calculul logaritmului discret

- Algoritmii non-generici sunt potențial mai puternici decât cei generici;
- Cel mai cunoscut algoritm pentru PLD în $\mathbb{Z}_p^*$ cu p prim este algoritmul GNFS (General Number Field Sieve) cu complexitate timp $2^{\mathcal{O}(n^{1/3} \cdot (\log n)^{2/3})}$ unde $|p| = \mathcal{O}(n)$;

Algoritmi non-generici pentru calculul logaritmului discret

- Algoritmii non-generici sunt potențial mai puternici decât cei generici;
- Cel mai cunoscut algoritm pentru PLD în $\mathbb{Z}_p^*$ cu p prim este algoritmul GNFS (General Number Field Sieve) cu complexitate timp $2^{\mathcal{O}(n^{1/3} \cdot (\log n)^{2/3})}$ unde $|p| = \mathcal{O}(n)$;
- Există și un alt algoritm non-generic numit *metoda de calcul a indicelui* care rezolvă DLP în grupuri ciclice $\mathbb{Z}_p^*$ cu p prim în timp sub-expoențial în lungimea lui p.

Algoritmi non-generici pentru calculul logaritmului discret

- Algoritmii non-generici sunt potențial mai puternici decât cei generici;
- Cel mai cunoscut algoritm pentru PLD în $\mathbb{Z}_p^*$ cu p prim este algoritmul GNFS (General Number Field Sieve) cu complexitate timp $2^{\mathcal{O}(n^{1/3}\cdot(\log n)^{2/3})}$ unde $|p| = \mathcal{O}(n)$;
- Există și un alt algoritm non-generic numit *metoda de calcul a indicelui* care rezolvă DLP în grupuri ciclice $\mathbb{Z}_p^*$ cu p prim în timp sub-expoențial în lungimea lui p.
- Această metodă seamănă cu algoritmul sitei pătratice pentru factorizare;

Algoritmi non-generici pentru calculul logaritmului discret

- Algoritmii non-generici sunt potențial mai puternici decât cei generici;
- Cel mai cunoscut algoritm pentru PLD în $\mathbb{Z}_p^*$ cu p prim este algoritmul GNFS (General Number Field Sieve) cu complexitate timp $2^{\mathcal{O}(n^{1/3} \cdot (\log n)^{2/3})}$ unde $|p| = \mathcal{O}(n)$;
- Există și un alt algoritm non-generic numit *metoda de calcul a indicelui* care rezolvă DLP în grupuri ciclice $\mathbb{Z}_p^*$ cu p prim în timp sub-expoențial în lungimea lui p.
- Această metodă seamănă cu algoritmul sitei pătratice pentru factorizare;
- Metoda funcționează în 2 etape; prima etapă este de pre-procesare și necesită cunoașterea modulului p și a bazei g;

Metoda de calcul a indicelui

- **Pasul 1**. Fie $q = p - 1$, ordinul lui $\mathbb{Z}_p^*$ și $B = \{p_1, ..., p_k\}$ o bază de numere prime mici;

Metoda de calcul a indicelui

- **Pasul 1**. Fie $q = p - 1$, ordinul lui $\mathbb{Z}_p^*$ și $B = \{p_1, ..., p_k\}$ o bază de numere prime mici;

- Se caută $l \geq k$ numere distincte $x_1, ..., x_l \in \mathbb{Z}_q$ pentru care $g_i = [g^{x_i} \bmod p]$ este "mic" așa încât toți factorii primi ai lui g_i se găsesc în B;

Metoda de calcul a indicelui

- **Pasul 1**. Fie $q = p - 1$, ordinul lui $\mathbb{Z}_p^*$ și $B = \{p_1, ..., p_k\}$ o bază de numere prime mici;
- Se caută $l \geq k$ numere distincte $x_1, ..., x_l \in \mathbb{Z}_q$ pentru care $g_i = [g^{x_i} \bmod p]$ este "mic" așa încât toți factorii primi ai lui g_i se găsesc în B;
- Vor rezulta relații de forma

$$g^{x_i} = p_1^{e_{i,1}} \cdot p_2^{e_{i,2}} \cdot \dots \cdot p_k^{e_{i,k}} \bmod p$$

unde $1 \leq i \leq k$.

Metoda de calcul a indicelui

- **Pasul 1**. Fie $q = p - 1$, ordinul lui $\mathbb{Z}_p^*$ și $B = \{p_1, ..., p_k\}$ o bază de numere prime mici;
- Se caută $l \geq k$ numere distincte $x_1, ..., x_l \in \mathbb{Z}_q$ pentru care $g_i = [g^{x_i} \text{ mod } p]$ este "mic" așa încât toți factorii primi ai lui g_i se găsesc în B;
- Vor rezulta relații de forma

$$g^{x_i} = p_1^{e_{i,1}} \cdot p_2^{e_{i,2}} \cdot \dots \cdot p_k^{e_{i,k}} \text{ mod } p$$

unde $1 \leq i \leq k$.

- sau

$$x_i = e_{i,1} \log_g p_1 \cdot e_{i,2} \log_g p_2 \cdot \dots \cdot e_{i,k} \log_g p_k \text{ mod } p - 1$$

Metoda de calcul a indicelui

- În ecuațiile de mai sus, necunoscutele sunt valorile $\{log_g p_i\}$

Metoda de calcul a indicelui

- În ecuațiile de mai sus, necunoscutele sunt valorile $\{log_g p_i\}$
- **Pasul 2**. Se dă y pentru care se caută $log_g y$;

Metoda de calcul a indicelui

- În ecuațiile de mai sus, necunoscutele sunt valorile $\{log_g p_i\}$
- **Pasul 2**. Se dă y pentru care se caută $log_g y$;
- Se găsește o valoare $s \in \mathbb{Z}_q$ pentru care $g^s \cdot y$ mod p este "mic" și poate fi factorizat peste baza B, obținându-se o relație de forma

$$g^s \cdot y = p_1^{s_1} \cdot p_2^{s_2} \cdot ... \cdot p_k^{s_k} \bmod p$$

Metoda de calcul a indicelui

- În ecuațiile de mai sus, necunoscutele sunt valorile $\{log_g p_i\}$
- **Pasul 2**. Se dă y pentru care se caută $log_g y$;
- Se găsește o valoare $s \in \mathbb{Z}_q$ pentru care $g^s \cdot y$ mod p este "mic" și poate fi factorizat peste baza B, obținându-se o relație de forma

$$g^s \cdot y = p_1^{s_1} \cdot p_2^{s_2} \cdot \ldots \cdot p_k^{s_k} \bmod p$$

- sau

$$s + \log_g y = s_1 \log_g p_1 \cdot s_2 \log_g p_2 \cdot \ldots \cdot s_k \log_g p_k \bmod p - 1$$

unde s și s_i se cunosc;

Metoda de calcul a indicelui

- În combinație cu ecuațiile din slide-ul anterior, sunt în total $l + 1 \geq k + 1$ ecuații liniare în $k + 1$ necunoscute $log_g p_i$, pentru $i = 1, ..., k$ și $log_g y$.

Metoda de calcul a indicelui

- În combinație cu ecuațiile din slide-ul anterior, sunt în total $l + 1 \geq k + 1$ ecuații liniare în $k + 1$ necunoscute $log_g p_i$, pentru $i = 1, ..., k$ și $log_g y$.
- O variantă optimizată a acestei metode rulează în timp $2^{\mathcal{O}(\sqrt{n \cdot \log n})}$ pentru un grup $\mathbb{Z}_p^*$ cu p prim de lungime n.

Metoda de calcul a indicelui

- În combinație cu ecuațiile din slide-ul anterior, sunt în total $l + 1 \geq k + 1$ ecuații liniare în $k + 1$ necunoscute $log_g p_i$, pentru $i = 1, ..., k$ și $log_g y$.
- O variantă optimizată a acestei metode rulează în timp $2^{\mathcal{O}(\sqrt{n \cdot \log n})}$ pentru un grup $\mathbb{Z}_p^*$ cu p prim de lungime n.
- Algoritmul este sub-exponențial în lungimea lui p.

Exemplu

- Fie p = 101, g = 3 și y=87. Se știe că
$$3^{10} = 65 \bmod 101 \text{ și } 65 = 5 \cdot 13$$
La fel,
$$3^{12} = 2^4 \cdot 5 \bmod 101$$
și
$$3^{14} = 13 \bmod 101$$

Exemplu

- Fie p = 101, g = 3 și y=87. Se știe că
$$3^{10} = 65 \bmod 101 \text{ și } 65 = 5 \cdot 13$$
La fel,
$$3^{12} = 2^4 \cdot 5 \bmod 101$$
și
$$3^{14} = 13 \bmod 101$$
- Prin urmare:
$$10 = log_3 5 + log_3 13 \bmod 100$$
$$12 = 4 \cdot log_3 2 + log_3 5 \bmod 100$$
$$14 = log_3 13 \bmod 100$$

Exemplu

- Fie p = 101, g = 3 și y=87. Se știe că
$$3^{10} = 65 \bmod 101 \text{ și } 65 = 5 \cdot 13$$
La fel,
$$3^{12} = 2^4 \cdot 5 \bmod 101$$
și
$$3^{14} = 13 \bmod 101$$
- Prin urmare:
$$10 = log_3 5 + log_3 13 \bmod 100$$
$$12 = 4 \cdot log_3 2 + log_3 5 \bmod 100$$
$$14 = log_3 13 \bmod 100$$
- Baza de numere prime mici este:
$$B = \{2, 5, 13\}$$

Metoda de calcul a indicelui

- De asemenea, $3^5 \cdot 87 = 32 = 2^5$ mod 101 sau

$$5 + log_3 87 = 5 \cdot log_3 2 \text{ mod } 100$$

Metoda de calcul a indicelui

- De asemenea, $3^5 \cdot 87 = 32 = 2^5 \bmod 101$ sau

$$5 + log_3 87 = 5 \cdot log_3 2 \bmod 100$$

- Combinând primele 3 ecuații, rezultă că $4 \cdot log_3 2 = 16 \bmod 100$;

Metoda de calcul a indicelui

- De asemenea, $3^5 \cdot 87 = 32 = 2^5 \bmod 101$ sau

$$5 + log_3 87 = 5 \cdot log_3 2 \bmod 100$$

- Combinând primele 3 ecuații, rezultă că $4 \cdot log_3 2 = 16 \bmod 100$;
- Această relație nu determină $log_3 2$ unic dar se găsesc 4 valori posibile: 4, 29, 54 și 79.

Metoda de calcul a indicelui

- De asemenea, $3^5 \cdot 87 = 32 = 2^5 \bmod 101$ sau

$$5 + log_3 87 = 5 \cdot log_3 2 \bmod 100$$

- Combinând primele 3 ecuații, rezultă că $4 \cdot log_3 2 = 16 \bmod 100$;
- Această relație nu determină $log_3 2$ unic dar se găsesc 4 valori posibile: 4, 29, 54 și 79.
- Prin încercări, se găsește $log_3 2 = 39$, și deci $log_3 87 = 40$.

Funcții hash rezistente la coliziuni

- În cadrul criptografiei simetrice, am văzut construcții euristice de funcții hash rezistente la coliziuni care sunt folosite pe larg în practică;

Funcții hash rezistente la coliziuni

- În cadrul criptografiei simetrice, am văzut construcții euristice de funcții hash rezistente la coliziuni care sunt folosite pe larg în practică;
- În continuare prezentăm o construcție pentru funcții hash rezistente la coliziuni bazată pe PLD (prezumpția logaritmului discret) în grupuri de ordin prim;

Funcții hash rezistente la coliziuni

- În cadrul criptografiei simetrice, am văzut construcții euristice de funcții hash rezistente la coliziuni care sunt folosite pe larg în practică;
- În continuare prezentăm o construcție pentru funcții hash rezistente la coliziuni bazată pe PLD (prezumpția logaritmului discret) în grupuri de ordin prim;
- Cosntrucția este mai puțin eficientă în practică ....

Funcții hash rezistente la coliziuni

- În cadrul criptografiei simetrice, am văzut construcții euristice de funcții hash rezistente la coliziuni care sunt folosite pe larg în practică;
- În continuare prezentăm o construcție pentru funcții hash rezistente la coliziuni bazată pe PLD (prezumpția logaritmului discret) în grupuri de ordin prim;
- Cosntrucția este mai puțin eficientă în practică ....
- ... însă arată că e posibil a obține rezistența la coliziuni pe baza unor prezumpții criptografice standard și bine studiate.

Funcții hash rezistente la coliziuni

- Fie $\mathbb{G}$ un grup ciclic de ordin prim q (cu $|q| = n$) și g un generator al său iar h un element aleator din $\mathbb{G}$;

Funcții hash rezistente la coliziuni

- Fie $\mathbb{G}$ un grup ciclic de ordin prim q (cu $|q| = n$) și g un generator al său iar h un element aleator din $\mathbb{G}$;
- Definim o funcție hash H cu intrarea de lungime fixă după cum urmează:

Funcții hash rezistente la coliziuni

- Fie $\mathbb{G}$ un grup ciclic de ordin prim q (cu $|q| = n$) și g un generator al său iar h un element aleator din $\mathbb{G}$;
- Definim o funcție hash H cu intrarea de lungime fixă după cum urmează:
- H: pentru intrarea $(x_1, x_2) \in \mathbb{Z}_q \times \mathbb{Z}_q$

$$H(x_1, x_2) = g^{x_1} h^{x_2}$$

Funcții hash rezistente la coliziuni

Teoremă

Dacă problema logaritmului discret este dificilă în grupul $\mathbb{G}$, atunci construcția de mai sus este o funcție hash de intrare fixă rezistentă la coliziuni.

Funcții hash rezistente la coliziuni

Teoremă

Dacă problema logaritmului discret este dificilă în grupul $\mathbb{G}$, atunci construcția de mai sus este o funcție hash de intrare fixă rezistentă la coliziuni.

- Pentru demonstrație vom folosi abordarea reducționistă:

Funcții hash rezistente la coliziuni

Teoremă

Dacă problema logaritmului discret este dificilă în grupul $\mathbb{G}$, atunci construcția de mai sus este o funcție hash de intrare fixă rezistentă la coliziuni.

- Pentru demonstrație vom folosi abordarea reducționistă:
- Arătăm că o construcție criptografică e sigură atâta timp cât problema pe care se bazează e dificilă, în felul următor:

Funcții hash rezistente la coliziuni

Teoremă

Dacă problema logaritmului discret este dificilă în grupul $\mathbb{G}$, atunci construcția de mai sus este o funcție hash de intrare fixă rezistentă la coliziuni.

- Pentru demonstrație vom folosi abordarea reducționistă:
- Arătăm că o construcție criptografică e sigură atâta timp cât problema pe care se bazează e dificilă, în felul următor:
- Prezentăm o reducție explicită arătând cum putem transforma un adversar eficient $\mathcal{A}$ care atacă securitatea construcției cu probabilitate ne-neglijabilă într-un algoritm eficient $\mathcal{A}'$ care rezolvă problema dificilă;

Demonstrație

- Fie H o funcție hash precum cea din construcția de mai sus și $\mathcal{A}$ un adversar PPT; notăm cu

$$\epsilon(n) = Pr[Hash - coll_{\mathcal{A},H} = 1]$$

probabilitatea ca $\mathcal{A}$ să găsească coliziuni în funcția H;

Demonstrație

- Fie H o funcție hash precum cea din construcția de mai sus și $\mathcal{A}$ un adversar PPT; notăm cu

$$\epsilon(n) = Pr[Hash - coll_{\mathcal{A},H} = 1]$$

  probabilitatea ca $\mathcal{A}$ să găsească coliziuni în funcția H;
- Arătăm că $\mathcal{A}$ poate fi folosit de $\mathcal{A}'$ pentru a rezolva DLP cu probabilitate de succes $\epsilon(n)$;

Demonstrație

- **Algoritmul** $\mathcal{A}'$ primește la intrare $s = (\mathbb{G}, q, g, h)$.

Demonstrație

- **Algoritmul** $\mathcal{A}'$ primește la intrare $s = (\mathbb{G}, q, g, h)$.
  1. Execută $\mathcal{A}(s)$ și obține x și x';

Demonstrație

- **Algoritmul** $\mathcal{A}'$ primește la intrare $s = (\mathbb{G}, q, g, h)$.
  1. Execută $\mathcal{A}(s)$ și obține x și x';
  2. Dacă $x \neq x'$ și $H(x) = H(x')$ atunci

Demonstrație

- **Algoritmul** $\mathcal{A}'$ primește la intrare $s = (\mathbb{G}, q, g, h)$.
  1. Execută $\mathcal{A}(s)$ și obține x și x';
  2. Dacă $x \neq x'$ și $H(x) = H(x')$ atunci
     2.1 Dacă $h = 1$ întoarce 0;

Demonstrație

- **Algoritmul** $\mathcal{A}'$ primește la intrare $s = (\mathbb{G}, q, g, h)$.
  1. Execută $\mathcal{A}(s)$ și obține x și x';
  2. Dacă $x \neq x'$ și $H(x) = H(x')$ atunci
     2.1 Dacă $h = 1$ întoarce 0;
     2.2 Altfel ($h \neq 1$), notează $x = (x_1, x_2)$ și $x' = (x'_1, x'_2)$. Întoarce $[(x_1 - x_1{}') \cdot (x'_2 - x_2)^{-1} \bmod q]$.

Demonstrație

- **Algoritmul** $\mathcal{A}'$ primește la intrare $s = (\mathbb{G}, q, g, h)$.
  1. Execută $\mathcal{A}(s)$ și obține x și x';
  2. Dacă $x \neq x'$ și $H(x) = H(x')$ atunci
     2.1 Dacă $h = 1$ întoarce 0;
     2.2 Altfel ($h \neq 1$), notează $x = (x_1, x_2)$ și $x' = (x'_1, x'_2)$. Întoarce $[(x_1 - x_1{}') \cdot (x'_2 - x_2)^{-1} \bmod q]$.
- Clar, $\mathcal{A}'$ rulează în timp polinomial;

Demonstrație

- **Algoritmul** $\mathcal{A}'$ primește la intrare $s = (\mathbb{G}, q, g, h)$.
  1. Execută $\mathcal{A}(s)$ și obține x și x';
  2. Dacă $x \neq x'$ și $H(x) = H(x')$ atunci
     2.1 Dacă $h = 1$ întoarce 0;
     2.2 Altfel ($h \neq 1$), notează $x = (x_1, x_2)$ și $x' = (x'_1, x'_2)$. Întoarce $[(x_1 - x_1{}') \cdot (x'_2 - x_2)^{-1} \bmod q]$.
- Clar, $\mathcal{A}'$ rulează în timp polinomial;
- Verificăm faptul că dacă $\mathcal{A}$ găsește o coliziune, $\mathcal{A}'$ întoarce răspunsul corect $log_g h$:

Demonstrație

- Dacă $h = 1$, atunci răspunsul lui $\mathcal{A}'$ este corect pentru că $log_g h = 0$;

Demonstrație

- Dacă $h = 1$, atunci răspunsul lui $\mathcal{A}'$ este corect pentru că $log_g h = 0$;

- Altfel, existența unei coliziuni implică:

$$\begin{aligned} H(x_1, x_2) = H(x_1', x_2') &\Rightarrow g^{x_1} h^{x_2} = g^{x_1'} h^{x_2'} \\ &\Rightarrow g^{x_1 - x_1'} = h^{x_2' - x_2} \end{aligned}$$

Demonstrație

- Dacă $h = 1$, atunci răspunsul lui $\mathcal{A}'$ este corect pentru că $log_g h = 0$;
- Altfel, existența unei coliziuni implică:

$$\begin{aligned} H(x_1, x_2) = H(x_1', x_2') &\Rightarrow g^{x_1} h^{x_2} = g^{x_1'} h^{x_2'} \\ &\Rightarrow g^{x_1 - x_1'} = h^{x_2' - x_2} \end{aligned}$$

- Notăm $\Delta = x_2' - x_2$.

Demonstrație

- Dacă $h = 1$, atunci răspunsul lui $\mathcal{A}'$ este corect pentru că $log_g h = 0$;
- Altfel, existența unei coliziuni implică:

$$\begin{aligned} H(x_1, x_2) = H(x_1', x_2') &\Rightarrow g^{x_1} h^{x_2} = g^{x_1'} h^{x_2'} \\ &\Rightarrow g^{x_1 - x_1'} = h^{x_2' - x_2} \end{aligned}$$

- Notăm $\Delta = x_2' - x_2$.
- Observăm că $\Delta \neq 0 \bmod q$. De ce?

Demonstrație

- Dacă $h = 1$, atunci răspunsul lui $\mathcal{A}'$ este corect pentru că $log_g h = 0$;

- Altfel, existența unei coliziuni implică:

$$\begin{aligned} H(x_1, x_2) = H(x_1', x_2') &\Rightarrow g^{x_1} h^{x_2} = g^{x_1'} h^{x_2'} \\ &\Rightarrow g^{x_1 - x_1'} = h^{x_2' - x_2} \end{aligned}$$

- Notăm $\Delta = x_2' - x_2$.

- Observăm că $\Delta \neq 0 \bmod q$. De ce?

- Pentru că ar însemna că $[(x_1 - x_1') \bmod q] = 0$ și deci $x = (x_1, x_2) = (x_1', x_2') = x'$, contradicție cu $x \neq x'$;

Demonstrație

- Cum q-prim și $\Delta \neq 0 \bmod q$ atunci inversul $[\Delta^{-1} \bmod q]$ există;

Demonstrație

- Cum q-prim și $\Delta \neq 0 \bmod q$ atunci inversul $[\Delta^{-1} \bmod q]$ există;
- Deci $g^{(x_1 - x_1')\cdot\Delta^{-1}} = (h^{x_2' - x_2})^{\Delta^{-1} \bmod q} = h^{\Delta\cdot\Delta^{-1} \bmod q} = h$

Demonstrație

- Cum q-prim și $\Delta \neq 0 \bmod q$ atunci inversul $[\Delta^{-1} \bmod q]$ există;
- Deci $g^{(x_1 - x_1') \cdot \Delta^{-1}} = (h^{x_2' - x_2})^{\Delta^{-1} \bmod q} = h^{\Delta \cdot \Delta^{-1} \bmod q} = h$
- rezultă că $log_g h = [(x_1 - x_1') \cdot \Delta^{-1} \bmod q] = [(x_1 - x_1') \cdot (x_2 - x_2')^{-1} \bmod q]$

Demonstrație

- Cum q-prim și $\Delta \neq 0 \bmod q$ atunci inversul $[\Delta^{-1} \bmod q]$ există;
- Deci $g^{(x_1 - x_1')\cdot\Delta^{-1}} = (h^{x_2' - x_2})^{\Delta^{-1} \bmod q} = h^{\Delta\cdot\Delta^{-1} \bmod q} = h$
- rezultă că $log_g h = [(x_1 - x_1')\cdot\Delta^{-1} \bmod q] = [(x_1 - x_1')\cdot(x_2 - x_2')^{-1} \bmod q]$
- Observăm că $\mathcal{A}'$ rezolvă DLP corect cu probabilitate exact $\epsilon(n)$

Demonstrație

- Cum q-prim și $\Delta \neq 0 \bmod q$ atunci inversul $[\Delta^{-1} \bmod q]$ există;
- Deci $g^{(x_1 - x'_1)\cdot \Delta^{-1}} = (h^{x'_2 - x_2})^{\Delta^{-1} \bmod q} = h^{\Delta \cdot \Delta^{-1} \bmod q} = h$
- rezultă că $log_g h = [(x_1 - x'_1)\cdot \Delta^{-1} \bmod q] = [(x_1 - x'_1)\cdot (x_2 - x'_2)^{-1} \bmod q]$
- Observăm că $\mathcal{A}'$ rezolvă DLP corect cu probabilitate exact $\epsilon(n)$
- Cum DLP este dificilă din ipoteză, concluzionăm că $\epsilon(n)$ este neglijabilă.

Important de reținut!

- Cel mai bun algoritm pentru DLP este sub-exponențial;
- Se pot construi funcții hash rezistente la coliziuni bazate pe dificultatea DLP;