Das Generalized Birthday Problem

Problem Birthday

Gegeben: L_1, L_2 Listen mit Elementen aus $\{0,1\}^n$
Gesucht: $x_1 \in L_1$ und $x_2 \in L_2$ mit $x_1 \oplus x_2 = \mathbf{0}$.

Anwendungen:

- Meet-in-the-Middle Angriffe (z.B. für RSA, ElGamal)
- Kollisionen für Hashfunktionen $h : \{0,1\}^* \to \{0,1\}^n$

Problem Generalized Birthday

Gegeben: $L_1, \ldots, L_k$ Listen mit Elementen aus $\{0,1\}^n$, unabhängig und gleichverteilt gezogen
Gesucht: $x_1 \in L_1, \ldots, x_k \in L_k$ mit $x_1 \oplus \ldots \oplus x_k = \mathbf{0}$

- Listen können auf beliebige Länge erweitert werden.
- Wir erwarten die Existenz einer Lösung sobald $|L_1| \cdot \ldots \cdot |L_k| > 2^n$.

Zusammenfügen zweier Listen

Definition Join-Operator

Wir bezeichnen mit $\mathrm{low}_\ell(x)$ die ℓ niederwertigsten Bits von x. Wir definieren für zwei Listen L_1, L_2 den Join-Operator

$$L_1 \bowtie_\ell L_2 = \{(x_1, x_2, x_1 \oplus x_2) \in L_1 \times L_2 \times \{0,1\}^n \mid \mathrm{low}_\ell(x_1) = \mathrm{low}_\ell(x_2)\}.$$

Eigenschaften:

- Es gilt $\mathrm{low}_\ell(x_1 \oplus x_2) = \mathbf{0}$ gdw $\mathrm{low}_\ell(x_1) = \mathrm{low}_\ell(x_2)$.
- Bei Eingabe L_1, L_2 kann $L_1 \bowtie L_2$ leicht berechnet werden.
- Falls $x_1 \oplus x_2 = x_3 \oplus x_4$, dann gilt $x_1 \oplus x_2 \oplus x_3 \oplus x_4 = \mathbf{0}$.
- Falls $\mathrm{low}_\ell(x_1 \oplus x_2) = \mathbf{0}$ und $\mathrm{low}_\ell(x_3 \oplus x_3) = \mathbf{0}$, dann gilt

$$\mathrm{low}_\ell(x_1 \oplus x_2 \oplus x_3 \oplus x_4) = \mathbf{0} \text{ und } \mathrm{Ws}[x_1 \oplus x_2 \oplus x_3 \oplus x_4 = \mathbf{0}] = \frac{1}{2^{n-\ell}}.$$

Algorithmus für das 4-Listen Problem

Algorithmus 4-Listen Problem

EINGABE: L_1, L_2, L_3, L_4 der Länge $|L_i| = 2^{\frac{n}{3}}$ mit Elementen aus $\{0,1\}^n$

1. Setze $\ell := \frac{n}{3}$.
2. Berechne $L_{12} = L_1 \bowtie_\ell L_2$ und $L_{34} = L_3 \bowtie_\ell L_4$.
3. Berechne $L_{1234} = L_{12} \bowtie_n L_{34}$.

AUSGABE: Elemente von L_{1234}

Korrektheit des 4-Listen Problem Algorithmus

Korrektheit:

- Elemente von L_{12}, L_{34} erfüllen $\text{low}_{\frac{n}{3}}(x_1 \oplus x_2) = \text{low}_{\frac{n}{3}}(x_3 \oplus x_4) = \mathbf{0}$.
- Wir erwarten Listenlänge
  $E[|L_{12}|] = \sum_{(x_1,x_2)\in L_1 \times L_2} \text{Ws}[\text{low}_{\frac{n}{3}}(x_1 \oplus x_2) = \mathbf{0}] = \frac{|L_1|\cdot|L_2|}{2^{\frac{n}{3}}} = 2^{\frac{n}{3}}$.
- Analog gilt $E[|L_{34}|] = 2^{\frac{n}{3}}$.
- Elemente von L_{1234} erfüllen $x_1 \oplus x_2 \oplus x_3 \oplus x_4 = \mathbf{0}$.
- Die erwartete Listenlänge $E[|L_{1234}|]$ von L_{1234} ist

$$\sum_{(x_1,\ldots,x_4)\in L_{12} \times L_{34}} \text{Ws}[x_1 \oplus \ldots \oplus x_4 = \mathbf{0} \mid \text{low}_{\frac{n}{3}}(x_1 \oplus x_2) = \text{low}_{\frac{n}{3}}(x_3 \oplus x_4)]$$
$$= \frac{E(|L_{12}|)\cdot E(|L_{34}|)}{2^{\frac{2n}{3}}} = 1.$$

- D.h. wir erwarten, dass L_{1234} mindestens eine Lösung enthält.

Laufzeitanalyse des 4-Listen Problem Algorithmus

Laufzeit und Speicherplatz:

- Die Listen $L_1, \ldots, L_4, L_{12}, L_{34}$ benötigen jeweils Platz $\tilde{\mathcal{O}}(2^{\frac{n}{3}})$.
- Die Konstruktion von L_{12}, L_{34} geht in Laufzeit $\tilde{\mathcal{O}}(2^{\frac{n}{3}})$.
- Konstruktion von L_{1234} benötigt ebenfalls Laufzeit $\tilde{\mathcal{O}}(2^{\frac{n}{3}})$.
- **Gesamt:** Zeit und Platz $\tilde{\mathcal{O}}(2^{\frac{n}{3}})$

Übungen: Modifizieren Sie den Algorithmus, so dass

- $\mathrm{low}_\ell(x_1 \oplus x_2) = \mathrm{low}_\ell(x_3 \oplus x_4) = c$ für ein $c \in \{0,1\}^\ell$.
- wir $x_1 \oplus x_2 \oplus x_3 \oplus x_4 = c'$ für ein $c' \in \{0,1\}^n$ lösen können.
- wir jede Instanz mit $k \geq 4$ in Zeit und Platz $\tilde{\mathcal{O}}(2^{\frac{n}{3}})$ lösen können.

4-Listen Problem in $\mathbb{Z}_{2^n}$

Ziel: Verwende Gruppe $(\mathbb{Z}_{2^n}, +)$ statt $(\{0,1\}^n, \oplus) = (\mathbb{F}_{2^n}, +)$.

Sei $-L = \{-x \in \mathbb{Z}_{2^n} \mid x \in L\}$.

Algorithmus 4-Listen Problem

EINGABE: L_1, L_2, L_3, L_4 mit Elementen aus $\{0,1\}^n$ der Länge $|L_i| = 2^{\frac{n}{3}}$

1. Setze $\ell := \frac{n}{3}$.
2. Berechne $L_{12} = L_1 \bowtie_\ell -L_2$ und $L_{34} = L_3 \bowtie_\ell -L_4$.
3. Berechne $L_{1234} = L_{12} \bowtie_n -L_{34}$.

AUSGABE: Elemente von L_{1234}

Korrektheit:

- Wir erhalten $(x_1, x_2, x_1 + x_2) \in L_{12}$ mit $x_1 + x_2 = 0 \bmod 2^\ell$.
- Man beachte: Für $x_1 + x_2 = 0 \bmod 2^\ell$ und $x_3 + x_4 = 0 \bmod 2^\ell$ gilt

$$x_1 + x_2 + x_3 + x_4 = 0 \bmod 2^\ell.$$

Algorithmus k-Listen Problem, $k = 2^m$

Algorithmus k-Listen Problem

EINGABE: $L_1, \ldots, L_{2^m}$ mit Elementen aus $\{0,1\}^n$, Länge $|L_i| = 2^{\frac{n}{m+1}}$

1. Setze $\ell := \frac{n}{m+1}$.
2. For $i := 1$ to $m-1$
   1. FOR $j := 1$ to 2^m step 2^i
      /* Join aller benachbarten Listen auf Level i des Baumes */
   2. Berechne $L_{j \ldots j+2^i-1} = L_{j \ldots j+2^{i-1}-1} \bowtie_\ell L_{j+2^{i-1} \ldots j+2^i-1}$.
3. Berechne $L_{1 \ldots 2^m} = L_{1 \ldots 2^{m-1}} \bowtie_n L_{2^{m-1}+1 \ldots 2^m}$.

AUSGABE: Elemente von $L_{1 \ldots 2^m}$

Beispiel für $k = 2^3$:

- Join für $i = 1$: $L_{12} = L_1 \bowtie_\ell L_2$, $L_{34} = L_3 \bowtie_\ell L_4$, $\ldots$, $L_{78} = L_7 \bowtie_\ell L_8$.
- Join für $i = 2$: $L_{1234} = L_{12} \bowtie_\ell L_{34}$, $L_{5678} = L_{56} \bowtie_\ell L_{78}$.
- Join in Schritt 3: $L_{1 \ldots 8} = L_{1 \ldots 4} \bowtie_n L_{5 \ldots 8}$.

Analyse des k-Listen Algorithmus

Korrektheit:

- Alle Startlisten besitzen Länge 2^ℓ.
- D.h. durch das Join auf unterster Ebene entstehen Listen mit erwarteter Länge $\frac{2^\ell \cdot 2^\ell}{2^\ell} = 2^\ell$.
- Damit entstehen in Schritt 2 stets Listen mit erwarteter Länge 2^ℓ.
- In Schritt 3 entsteht eine Liste $L_{1\ldots k}$ mit erwarteter Länge

$$\sum_{(x_1,\ldots,x_k)} \mathrm{Ws}[x_1 \oplus \ldots \oplus x_k = \mathbf{0} \mid \mathrm{low}_{(m-1)\ell}(x_1 \oplus \ldots \oplus x_{\frac{k}{2}}) = \mathrm{low}_{(m-1)\ell}(x_{\frac{k}{2}+1} \oplus x_k)] = \frac{2^{2\ell}}{2^{n-(m-1)\ell}} = 1.$$

Analyse des k-Listen Algorithmus

Laufzeit und Platz:

- Die Listen $L_1, \ldots, L_k$ besitzen benötigen jeweils Platz $\tilde{\mathcal{O}}(2^\ell)$.
- In Schritt 2 berechnen wir $k-2$ Listen mit erwarteter Länge $\tilde{\mathcal{O}}(2^\ell)$.
- Damit erhalten wir Speicherplatzbedarf $\tilde{\mathcal{O}}(k2^\ell) = \tilde{\mathcal{O}}(k2^{\frac{n}{\log k+1}})$.
- Die Laufzeit für alle $k-1$ Join-Operationen beträgt $\tilde{\mathcal{O}}(2^\ell)$.
- Damit ist die Gesamtlaufzeit ebenfalls $\tilde{\mathcal{O}}(k2^\ell) = \tilde{\mathcal{O}}(k2^{\frac{n}{\log k+1}})$
- Für $k = 2^{\sqrt{n}}$ erhalten wir Zeit und Speicherplatz Komplexität

$$\tilde{\mathcal{O}}(2^{\sqrt{n}} \cdot 2^{\frac{n}{\sqrt{n}+1}}) = \tilde{\mathcal{O}}(2^{2\sqrt{n}})$$

- Dies ist eine subexponentielle Funktion in n.

Übung: Konstruieren Sie einen Algorithmus für $k = 2^m + j, 0 < j < 2^m$ mit Komplexität $\tilde{\mathcal{O}}(k2^{\frac{n}{\log k+1}})$.

Offenes Problem:
Geht es für $k = 2^m + j$ besser? Für $k = 3$ besser als $\tilde{\mathcal{O}}(2^{\frac{n}{2}})$?

Urbild Angriff auf Inkrementelle Hashfunktionen

AdHash Konstruktion: (Bellare, Micciancio 1997)

- Hashe Nachricht $x = (x_1, \ldots, x_k)$ als
$$H(x) = \sum_{i=1}^{k} h(i, x_i) \bmod M.$$
- **Inkrementell:** Block x_i kann leicht durch x_i' ersetzt werden.
- NASD Instantiierung: $M = 2^{256}$

Algorithmus: Urbild Angriff auf AdHash

EINGABE: Modul $M = 2^{256}$, Hashwert c

1. Generiere Listen $L_1, \ldots, L_k$ mit $|L_i| = 2^{\frac{n}{\log k+1}}$.
2. Liste L_i enthält $y_j^{(i)} = h(i, x_j)$ für zufällig gewählte x_j.
3. k-Listen Algorithmus liefert $y_{j_1}^{(1)}, \ldots, y_{j_k}^{(k)}$ mit
$$y_{j_1}^{(1)} + \ldots + y_{j_k}^{(k)} = c \bmod 2^{256} \text{ und } y_{j_i}^{(i)} = h(i, x_{j_i}).$$

AUSGABE: $x = (x_{j_1}, \ldots, x_{j_k})$ mit $H(x) = c \bmod M$

Urbild Angriff auf Inkrementelle Hashfunktionen

Komplexität:

- Naive Urbildberechnung benötigt erwartet 2^{256} H-Auswertungen.
- Für unseren Angriff ist der k-Listen Algorithmus laufzeitbestimmend.
- Auswerten von $k \cdot 2^{\frac{n}{\log k+1}}$ für $k = 128$ liefert $2^7 \cdot 2^{32} = 2^{39}$.
- Allgemein: Erhalten einen Angriff mit Komplexität $\tilde{\mathcal{O}}(2^{2\sqrt{\log M}})$.
- D.h. für 80-Bit Sicherheit muss $M > 2^{1600}$ gewählt werden.

Fälschen von einfachen Ringsignaturen

Idee: Ringsignatur

- Sei $U = \{u_1, \ldots, u_k\}$ eine Menge von Usern.
- Ein User u_i möchte eine Unterschrift im Namen von U leisten.
- Eine Ringsignatur schützt die Anonymität von u_i in U.

Ringsignatur von Back (1997)

Sei H eine Hashfunktion.

1. **Gen:** Generiere RSA Schlüssel (N_i, e_i, d_i) für alle User u_i.
2. **Sign:** User u_i wählt $m_j \in_R \mathbb{Z}_{N_j}$, $j \neq i$, Nachricht m, und berechnet

$$m_i = \left(H(m) \oplus \bigoplus_{j \neq i} (m_j^{e_j} \bmod N_j)\right)^{d_i} \bmod N_i.$$

   Ausgabe von (m, σ) mit der Signatur $\sigma = (m_1, \ldots, m_k)$.

3. **Vrfy:** Prüfe für (m, σ) die Identität

$$\bigoplus_{i=1}^{k} (m_i^{e_i} \bmod N_i) \stackrel{?}{=} H(m).$$

Fälschen von Ringsignaturen

Algorithmus Universelles Fälschen von Ringsignaturen

EINGABE: Nachricht m, (N_i, e_i) für $i = 1, \ldots, k$

1. Berechne Listen L_i für $i = 1, \ldots, k$ mit Elementen
$$x_j^{(i)} = m_j^{e_i} \bmod N_i \text{ für } m_j \in_R \mathbb{Z}_{N_i}.$$
2. k-Listen Algorithmus liefert $x_{j_1}^{(1)}, \ldots, x_{j_n}^{(n)}$ mit
$$x_{j_1}^{(1)} \oplus \ldots \oplus x_{j_n}^{(n)} = H(m).$$

AUSGABE: (m, σ) mit $\sigma = (m_{j_1}, \ldots, m_{j_n})$.

Komplexität:

- Sei $N = \max_i\{N_i\}$. Wir erhalten Komplexität $\mathcal{O}(k \cdot 2^{\frac{\log N}{\log k+1}})$.
- D.h. für $k = \theta(\log N)$ erhalten wir einen polynomiellen Angriff.

Polynomielle Vielfache mit kleinem Gewicht

Definition Gewicht eines Polynoms

Sei $p(x) = \sum_{i=0}^{n} p_i x^i \in \mathbb{F}_2[x]$. Das *Gewicht* $w(p)$ von $p(x)$ ist definiert als das Hamminggewicht des Koeffizientenvektors von $p(x)$, d.h.

$$w(p) = w((p_0, \ldots, p_n)).$$

- Betrachten Korrelationsattacken auf Stromchiffren.
- Diese benötigen Polynomvielfache sehr kleinen Gewichts.

Problem Polynomvielfache mit kleinem Gewicht

Gegeben: $p(x) \in \mathbb{F}_2[x]$ irreduzibel vom Grad n,
Gradschranke $d > n$, Gewicht k

Gesucht: $m(x) \in \mathbb{F}_2[x]$ mit $p(x) \mid m(x)$, Grad $\leq d$ und $w(m) \leq k$.

Konstruktion von Polynomvielfachen

Wir identifizieren Polynome in $\mathbb{F}_2[x]$ mit ihren Koeffizientenvektoren.

Algorithmus Polynomvielfache

EINGABE: $p(x) \in \mathbb{F}_2[x]$, Gewicht k

1. Setze die Gradschranke $d := 2^{\frac{n}{\log k+1}}$
2. Generiere Listen L_i, $i = 1, \ldots, k$ mit Elementen der Form
$$y_j^{(i)} = x^{a_j} \bmod p(x) \text{ für zufällig gewählte } a_j \leq d.$$
3. k-Listen Algorithmus liefert $y_{j_1}^{(1)}, \ldots, y_{j_k}^{(k)}$ mit
$$y_{j_1}^{(1)} \oplus \ldots \oplus y_{j_k}^{(k)} = \mathbf{0}.$$

AUSGABE: $m(x) = x^{a_{j_1}} + \ldots + x^{a_{j_k}}$

Konstruktion von Polynomvielfachen

Korrektheit:

- Wir definieren $\mathbb{F}_{2^n} = \mathbb{F}_2[x]/p(x)$. Addition zweier Polynome in $\mathbb{F}_{2^n}$ entspricht dem XOR ihrer Koeffizientenvektoren.
- Damit gilt $m(x) = x^{a_{j_1}} + \ldots + x^{a_{j_k}} = 0$ in $\mathbb{F}_{2^n}$, d.h. $p(x)$ teilt $m(x)$.
- Wegen $a_j \leq d$ besitzt $m(x)$ Grad höchstens d.
- Ferner besteht $m(x)$ aus k Monomen, d.h. besitzt Gewicht k.
- Für die Listengröße im k-Listen Alg. benötigen wir $d = 2^{\frac{n}{\log k+1}}$.
- D.h. unser Algorithmus funktioniert nur für hinreichend großes d.

Komplexität:

- Der k-Listen Algorithmus liefert Komplexität $\tilde{\mathcal{O}}(k \cdot 2^{\frac{n}{\log k+1}})$.
- Bsp.: $\mathrm{grad}(p) = 120$ und wir suchen Vielfaches mit Gewicht $k = 4$.
- Wir wählen $d = 2^{\frac{n}{\log k+1}} = 2^{\frac{120}{3}} = 2^{40}$ erhalten $k \cdot 2^{\frac{n}{\log k+1}} = 2^{42}$.

k-Listen Problem über $\mathbb{F}_{2^n}$ für $k \geq n$

Problem Generalized Birthday für $k \geq n$

Gegeben: $L_1, \ldots, L_k$ mit Elementen aus $\{0,1\}^n$, $|L_i| \geq 2$, $k \geq n$.
Gesucht: $\mathbf{x}_1 \in L_1, \ldots, \mathbf{x}_k \in L_k$ mit $\mathbf{x}_1 \oplus \ldots \oplus \mathbf{x}_k = \mathbf{0}$

Idee: (Algorithmus von Bellare, Micciancio 1997)

- ObdA $L_i = \{\mathbf{x}_{i,0}, \mathbf{x}_{i,1}\}$ für alle i, sonst entferne Elemente aus L_i.
- Definiere $b_i = \begin{cases} 0 & \text{falls } \mathbf{x}_{i,0} \text{ in } L_i \text{ ausgewählt wird.} \\ 1 & \text{falls } \mathbf{x}_{i,1} \text{ in } L_i \text{ ausgewählt wird.} \end{cases}$
- D.h. wird müssen $\mathbf{b} = (b_1, \ldots, b_n) \in \{0,1\}^n$ finden mit

$$b_1\mathbf{x}_{1,1} + (1-b_1)\mathbf{x}_{1,0} + \ldots + b_n\mathbf{x}_{n,1} + (1-b_n)\mathbf{x}_{n,0} = \mathbf{0}$$
$$\Leftrightarrow \quad b_1(\mathbf{x}_{1,1} - \mathbf{x}_{1,0}) + \ldots + b_n(\mathbf{x}_{n,1} - \mathbf{x}_{n,0}) = -(\mathbf{x}_1 + \ldots + \mathbf{x}_n)$$

- Dies ist ein lineares Gleichungssystem in den b_i.
- Falls die Matrix definiert durch die Vektoren $\mathbf{x}_{i,1} - \mathbf{x}_{i,0}$ vollen Rang besitzt, so können wir das System in Zeit $\mathcal{O}(n^3 + kn)$ lösen.