Privatly computing Set-Intersection using Bloom Filters

December 14, 2017

Niklas Jobst

TCS - Universität zu Lübeck

Motivation

⇒ Ziel dieses Projektes ist es verschiedene Methoden vergleichen, mit denen es möglich ist privacy preserving die Übereinstimmung der DNA einer Person zu einer anderen bzw. zu einem Referenz Exom zu ermitteln

⇒ Hierzu vergleichen wir die SNP Profile dieser Exome. (Unter Umständen wären auch Satelliten-Repeats möglich)

⇒ Viele Therapien in der Personalisierten Medizin sind an bestimmte SNP Profile gekoppelt.

⇒ Durch einen solchen Vergleich wäre es möglich festzustellen, ob für einen Patienten ein bestimmtes Medikament wirksam wäre oder nicht.

⇒ Die Grundlage aller dieser Methoden bilden sogenannte Bloom Filter

Bloom Filter

⇒ Technik um festzustellen, ob bestimmte Daten in einem Datensatz vorhanden sind oder nicht.

⇒ Aufbau:
- Ein mit Nullen initialisiertem m Bit langes Array
- k Hashfunktionen

⇒ Initialisierung
- Auf jedes Element des Datensatzes werden alle k Hashfunktionen angewendet.
- Jede der Hashfunktionen bildet auf das Array ab
- Die Positionen im Array die den Ausgaben entsprechen werden auf 1 gesetzt.

Bloom Filter

⇒ Datenprüfung:
- Auf ein zu überprüfendes Datenelement d werden wieder alle k Hashfunktionen angewendet
- Nur wenn alle Positionen im BF an den Punkten der Ausgabe 1 sind wird angenommen, dass sich d im Datensatz befindet

Algorithmus basierend auf Elgamal - In a Nutshell

Client

⇒ Erstellt Bloom Filter der Daten

⇒ Verschlüsselt jede Stelle des Bloom Filters mittels ElGamal

$$(R_i, S_i) = (g^{r_i}, pk^{r_i} * g^{1\text{-BF}_1[i]})$$

$[pk, (R_i, S_i)]$ →

← (V, W)

⇒ Client entschlüsselt mit sk Ciphertext von Server

⇒ Bestimmt Anzahl der Einträge an denen beide Bloom Filter null sind

⇒ Schätzt hieraus die Gesamtmenge an SNPs

Server

⇒ Erstellt Bloom Filter der Daten

⇒ Selektiert jene Stellen Stellen in dem BF die den Eintrag null besitzen.

⇒ Multipliziert an diesen Stellen die Werte des Ciphertextes vom Client auf

⇒ Rerandomisiert die entstandenen Ergebnisse

$$V = (g^s * \Pi_{i:BF_2[i]=0} R_i)$$

$$W = (pk^s * \Pi_{i:BF_2[i]=0} S_i)$$

Algorithmus - Step 1

$$(R_i, S_i) = (g^{\mathrm{r_i}}, pk^{\mathrm{r_i}} * g^{\text{1-BF}_1[\mathrm{i}]})$$

$$S_\mathrm{i} = pk^{r_i} \ * \begin{cases} g^0 = 1 \ bei \ BF_1[i] = 1 \\ g^1 = g \ bei \ BF_1[i] = 0 \end{cases}$$

Legende

pk: public key, sk: private key, g: Generator r_i: Zufallszahlen aus Z_q

Algorithmus - Step 2

$\Rightarrow$ Aufmultiplikation von R_i bzw S_i an jenen Stellen, an welchen $BF_2 = 0$ ist.

$\Rightarrow$ Rerandomisierung mit g^s bzw. pk^s

$\Rightarrow$ Diese Berechnungen sind ohne Datenverlust aufgrund der Homomorphie Eigenschaft von Elgamal möglich

$$V = (g^s * \Pi_{i:BF_2[i]=0} R_{i[j]})$$

$$W = (pk^s * \Pi_{i:BF_2[i]=0} S_i)$$

Legende

pk: Public key, sk: private key, g: Generator s: Zufallszahl aus Z_q

$\Rightarrow$ R_i, S_i aus vorherigem Schritt in V,W einsetzen.

$$V = (g^{s+r_{i_1}+r_{i_2}+ \ ... \ +r_{i_k}})$$

$$W = \begin{cases} pk^{s+r_{i_1}+r_{i_2}+ \ ... \ +r_{i_l}} * 1 \ falls \ BF_1 = 1, BF_2 = 0 \\ pk^{s+r_{i_1}+r_{i_2}+ \ ... \ +r_{i_m}} * g^x \ falls \ BF_1 = BF_2 = 0 \end{cases}$$

$$W = (pk^{s+r_{i_1}+r_{i_2}+ \ ... \ +r_{i_k}} * g^x)$$

Algorithmus - Step 3

$$\Sigma = W * V^{-sk}$$

$\Rightarrow$ V, W aus vorherigem Schritt einsetzen und für $pk = g^{sk}$

$$\Sigma = (g^{sk*s+r_{i_1}+r_{i_2}+ \ \dots \ +r_{i_k}} \ * \ g^{-sk*s+r_{i_1}+r_{i_2}+ \ \dots \ +r_{i_k}} \ * \ g^{x})$$

$$\Sigma = g^{x}$$

$\Rightarrow$ Durch wiederholtes dividieren durch g lässt sich x berechnen.

$\Rightarrow$ x entspricht der Anzahl der Stellen, an welchen beide Bloom Filter null sind

$\Rightarrow$ Dann berechnen sich die Stellen, an denen zumindest einer der BF eine 1 hat $z = m - x$

$\Rightarrow$ Hieraus lässt nun wiederum auf die Gesamtgröße der Vereinigung der Bloom Filter schließen.

$$|X| = \frac{ln(\frac{z}{m})}{k * \ ln(1 - \frac{1}{m})}$$

Herleitung der Abschätzung

$\Rightarrow$ Wahrscheinlichkeit, dass ein Bloomfilterbit Null ist
$z' = (1 - \frac{1}{m})^{k^* x}$

$\Rightarrow$ Mit der Abschätzung $(1 - \frac{t}{k})^k \approx e^{-t}$ gilt :

$$(1 - \frac{1}{m})^{k*x} = (1 - \frac{(k*x/m)}{k*x})^{k*x} \approx e^{\frac{-k*x}{m}}$$

$\Rightarrow$ Da binomialverteielt ist der Erwartungswert:
$z = m * (1 - \frac{1}{m})^{k^* x} \approx m * e^{\frac{-k*x}{m}}$

$\Rightarrow$ Durch Umformung nach x erhält man: $|X| = \frac{ln(\frac{z}{m})}{k*ln(1-\frac{1}{m})}$

Algorithmus - Anpassung

⇒ Durch den Algorithmus wurde Set-Union berechnet, wir benötigen jedoch die Set-Intersection,da wir die Gemeinsamkeiten der DNAs bestimmen wollen.

⇒ Hierzu invertieren wir zu Beginn des Algorithmus die Bloom Filter der Teilnehmer.

Paillier - Verfahren

Schlüsselerzeugung:

$\Rightarrow$ Client wählt zwei Primzahlen p,q , mit $ggt(pq, (p-1)(q-1)) = 1 und n = pq$

$\Rightarrow$ Der Generator g so gewählt, sodass $g \in (\mathbb{Z}n^2\mathbb{Z})$ und n die Ordnung von g teilt.

$\Rightarrow$ Secrect key: $\lambda = kgV(p-1, q-1)$

$\Rightarrow$ Public Key: (n, g)

Verschlüsselung:

$\Rightarrow$ Nachricht $m \in \mathbb{Z}_n^*$

$\Rightarrow$ Client wählt Zufallszahl $r \in \mathbb{Z}_{n^2}^*$

$\Rightarrow$ Ciphertext $c = g^m * r^n \mod n^2$

Entschlüsselung:

$\Rightarrow$ Benötigt zunächst $L(x) = \frac{(x-1)}{n}$

$\Rightarrow$ Plaintext $m = \frac{L(c^\lambda \ mod \ n^2)}{L(g^\lambda \ mod \ n^2)} \ mod \ n$

Homomorphie: Paillier ist homomorph gegenüber der Addition.

$$E(m_1 + m_2) = (E(m_1) * E(m_2))$$

Algorithmus basierend auf Paillier - In a Nutshell

Client

⇒ Erstellt Bloomfilter der Daten und invertiert jede Stelle des Bloomfilters.

⇒ Verschlüsselt jede Stelle des Bloomfilters mittels Paillier

$$c_i = (g^{IBF[i]} * r^n) \ mod \ n^2$$

$[pk, c]$ →

← $rerand(e_{1-j})$

⇒ Client entschlüsselt mit sk Ciphertexte von Server

⇒ Anzahl der entschlüsselten Nullen entspricht der Anzahl der sich überschneidenden Elemente

Server

⇒ Erstellt für jedes Element des Datensatzes einen Bloomfilter seiner Daten

⇒ Selektiert in jedem Blommfilter jene Stellen die den Eintrag Eins besitzen.

⇒ Multipliziert an diesen Stellen die Werte des Ciphertextes des Clients auf

⇒ Rerandomisiert die entstandenen Ergebnisse mit verschlüsselter Null

$$Rerand \ e_j = (ej * enc_{pailier}(0))$$

Algorithmus - Step 1

$$c_i = (g^{\mathrm{IBF[i]}} * {r_i}^{\mathrm{n}})$$

$$C_{\mathrm{i}} = r_i^n \ * \begin{cases} g^0 = 1 \ bei \ BF_1[i] = 1 \\ g^1 = g \ bei \ BF_1[i] = 0 \end{cases}$$

Legende

pk: public key, sk: private key, g: Generator r_i: Zufallszahlen aus Z_q

Algorithmus - Step 2

⇒ Für jedes Element des Datensatzes des Servers wird ein Bloomfilter erstellt.

⇒ Aufmultiplikation von c_i an jenen Stellen, an welchen $BF_{s[j]} = 1$ ist.

$$V_j = (g^{IBF_{i_1} + IBF_{i_2} + \ ... \ + IBF_{i_k}} \ * \ r_{i_1}^n \ * \ r_{i_2}^n \ * \ ... \ * \ r_{i_k}^n)$$

$$V_j = \ r_{i_1}^n * r_{i_2}^n * ... * r_{i_k}^n \begin{cases} g^{1+1+1+ \ ... \ +1} & wenn \ BF_c = 0, BF_{s[j]} = 1 \\ g^{0+0+0+ \ ... \ +0} & wenn \ BF_c = BF_{s[j]} = 1 \end{cases}$$

Legende

pk: Public key, sk: private key, g: Generator s: Zufallszahl aus Z_q

$\Rightarrow$ R_i, S_i aus vorherigem Schritt in V,W einsetzen.

$\Rightarrow$ Rerandomisierung mit g^s bzw. pk^s

$\Rightarrow$ Diese Berechnungen sind ohne Datenverlust aufgrund der Homomorphie Eigenschaft von Elgamal möglich

$$V = (g^s * \Pi_{i:BF_2[i]=0} R_i)$$

Algorithmus - Step 3

$$\Sigma = W * V^{-sk}$$

$\Rightarrow$ V, W aus vorherigem Schritt einsetzen und für $pk = g^{sk}$

$$\Sigma = (g^{sk*s+r_{i_1}+r_{i_2}+ \ \dots \ +r_{i_k}} \ * \ g^{-sk*s+r_{i_1}+r_{i_2}+ \ \dots \ +r_{i_k}} \ * \ g^{x})$$

$$\Sigma = g^{x}$$

Beispiel

Sei $X_{client} = rs12323, rs23453, rs34564$ und $X_{server} = rs98787, rs87676, rs76565$. Der Bloomfilter BF sei definiert mit dem Array $m = 0,0,0,0,0,0,0,0,0,0,0,0$ und den Hashfunktionen k_{1-3}. Zunächst wendet der Client die Hashfunktionen auf alle SNPs seines Datensatzes an: $k_{1(rs12323)} = 1, k_{2(rs12323)} = 2, k_{3(rs12323)} = 5; k_{1(rs23453)} = 2, k_{2(rs23453)} = 7, k_{3(rs23453)} = 0;$

Ergebnisse - Elgamal

⇒ Dauer für Vergleich des gesamten Exomes bei wenigen Minuten.

⇒ Laufzeit Unabhängig davon wie stark die Überschneidung zwischen zwischen den Datensätzen ist.

Überschneidung	14000	7500	5000	2000
Runtime (sec)	221	247	211	222
Abw. zur Überschn.	0.01%	3.3%	8.8%	36.8%

Table 1: Hashfunktionen : 14, Anzahl Bloomfilter Bits:3029660, Größe der Datensätze: 15000 SNPs

Array	1442696	1009887	577079	144270
Runtime (sec)	108	83	47	11
Abweichung	4%	6%	13%	51%

Table 2: Datensatz 1000 SNPs, Überschneidung 100, Hashfunktionen: 10

$\Rightarrow$ Die Laufzeit ist linear abhängig zur Anzahl der Bloomfilterbits

$\Rightarrow$ Die Stärke der Abweichung ist ebenfalls linear abhängig zur Anzahl der Bloomfilter Bits

Hashf.	1	4	7	10	14
Runtime (sec)	7	27	44	62	104
Abweichung	11%	13%	10%	9%	9%

Table 3: Datensatz 1000 SNPs, Überschneidung 100, Array: 504944

$\Rightarrow$ Anzahl der Hashfunktionen hat deutlich weniger Einfluss, jedoch kommt es bei hoher Anzahl zu vermehrt Falsch positiven Ergebnissen.

Ergebnisse-Paillier

Array	14139	12119	10099	8080
Runtime (sec)	219	194	183	163
Abweichung	1%	4%	6%	24%

Table 4: Hashf.7, Überschneidung 100, SNPs 1000

$\Rightarrow$ Paillier deutlich langsamer als Elgamal

$\Rightarrow$ Benötigt deutlich kleinere Bloomfilter für selbe Genauigkeit, jedoch ist die Bitweise Verschlüsselung sehr langsam

Array	141385	100989	75742
Runtime (sec)	2420	2318	2007
Abweichung	1%	4%	13%

Table 5: Hashf.7, Überschneidung 7500, SNPs 15000

$\Rightarrow$ Zum Vergleich von gesamten Exomen ca. 40 min

Vergleich

Abweichung	0.1%	0.6%	2%	3%	4%	6%
Runtime elgamal	467	150	17	15	11	6
Runtime paillier	510	340	150	150	135	120

Table 6: Hashf.7, Überschneidung 100, SNPs 1000