Privatly computing Set-Intersection using Bloom Filters

November 22, 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 - In a Nutshell

Alice

⇒ Erstellt Bloom Filter ihrer Daten

⇒ Verschlüsselt jede Stelle ihres Bloom Filters mittels ElGamal

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

$[pk, (R_i, S_i)]$ →

⇒ Alice entschlüsselt mit sk Ciphertext von Bob

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

⇒ Berechnet die Set-Union der BF

Bob

⇒ Erstellt Bloom Filter seiner Daten

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

⇒ Multipliziert an diesen Stellen die Werte des Ciphertextes von Alice auf

← (V, W)

⇒ 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

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

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

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.

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

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.

Implementierung - Wahl der Sprache

⇒ Einige Sprachen bieten Umfangreiche Umsetzung von ElGamal als vorgefertigtes Paket [Crypto++ (C++), pycrypto(Python)]

⇒ Allerdings sind diese sehr unübersichtlich und Zwischenschritte nicht einsehbar

⇒ "Sandbox" ähnliche Implementierungen zb. für Java vorhanden.

Implementierung - Wahl der Daten

⇒ Identifizierung von jedem möglichen SNP über Position und Basenaustausch möglich

⇒ Für unseren Anwendungsfall werden jedoch nur häufige SNPs gesucht ($> 5\%$) da nur diese in der personalisierten Medizin Verwendung finden.

⇒ Diese sind in einer Datenbank hinterlegt (dbSNP)