Private Set Operations from Oblivious Switching

19. Mai 2021

Inhaltsverzeichnis

1 Einleitung

Private Set Intersection, kurz PSI, beschreibt das Aufdecken (reveal) einer Schnittmenge (intersection) aus zwei Mengen, die Sets genannt werden. Abbildung 1 zeigt ein theoretisches Beispiel mit den Parteien Alice und Bob für das Berechnen der Intersection.

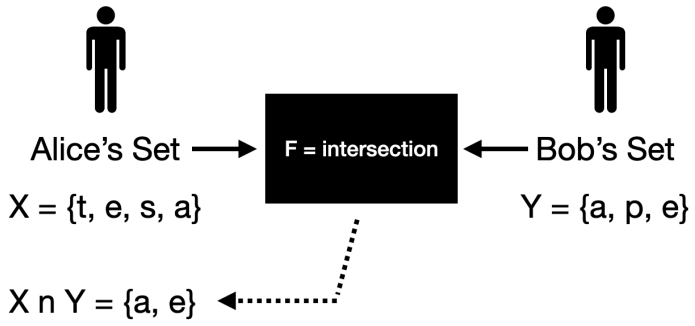


Abbildung 1: Beispiel Private Set Intersection (PSI)

Alice besitzt ein Set mit den Buchstaben (Strings) t,e,s und a. Bob besitzt ein Set mit den Buchstaben a, p und e. Wir erwarten, dass Alice die Buchstaben e und a lernt. Alice lernt nicht mehr als diese Buchstaben und kennt kein weiteres Element aus Bob's Set und Bob kennt ebenfalls kein weiteres Element aus Alice's Set. In PSI deckt das Ergebnis alle Inhalte der Intersection auf.

Aber was wäre, wenn ALice und Bob nur partielle Informationen der Intersection lernen wollen? Es existieren nur weniger Protokolle, die aktuell diese Problematik effizient lösen. Wir benötigen ein Protokoll, das eine beliebige Funktion f auf die Schnittmenge $X \oplus Y$ anwenden kann, ohne dabei die Items in den Sets selber zu verraten. Dieser Anwendungsfall wird Private Computation on Set Intersection (PCSI) genannt.

Dieses Dokument beschäftigt sich mit einem neu entwickeltem Protokoll namens **Private Set Operations from Oblivious Switching** , das auch als PSI Protokoll eingesetzt werden kann. Der Einfachheit halber kürzen wir das neue Protokoll mit P_{PSOFOS} ab. Mit P_{PSOFOS} ist es möglich, beliebige Funktionen auf die Schnittmenge von Alice und Bobs Sets anzuwenden.
Zusätzlich kann die neue Technik auch für private computation of union (PSU = Berechnung der Vereinigungsmenge zweier Sets) genutzt werden und darauf basierend auch für die Private ID Funktionalität. P_{PSOFOS} ist 2-2.5 x schneller als herkömmliche Standards und ist sicher gegenüber semi-honest adversaries in einem two-party setting (vgl. [1])

Die aktuell schnellsten PSI Protokolle basieren auf einem Ansatz von Benny Pinkas. Pinka's Ansatz (PSTY19) setze erstmals auf sog. OT-Extensions. Wir betrachten eine OT-Extension erstmal als eine Blackbox, die im Abschnitt 2.1 genauer erklärt wird. Ein Grundbaustein von P_{PSOFOS} sind ebenfalls OT-extension basierte PSI Techniken.

2 Grundlagen

2.1 Oblivious Transfer

Oblivious Transfer (OT) ist ein Protokoll, bei dem ein Absender eine von möglicherweise vielen Informationen an einen Empfänger überträgt, jedoch nicht weiß, welche Teile gelesen wurden. In der ersten Variante hat Alice (Sender) zwei Nachrichten m_0 und m_1, und Bob (Empfänger) hat ein Bit b. Bob möchte m_b empfangen, ohne dass Alice b lernt, während Alice sicherstellen möchte, dass Bob nur eine der beiden Nachrichten empfängt.

Ein einzelner Oblivious Transfer braucht public-key Operationen und ist deswegen als verhältnismäßig teuer zu betrachten. OT-Extension ist eine Technik, um die Laufzeit zu verbessern. Die OT-Extension erlaubt es, einen der Parteien n OT's auszuführen, während die Laufzeit bei public-key Operationen $O(k)$ und die Laufzeit bei fast symmetric-key Operations $O(n)$ ist.

2.2 Batch Oblivious PRF

Batched Oblivious PRF (OPRF) basiert auf OT-Extension (vgl. 2.1) und stellt einen Stapel batch genannt von oblivious PRF Instanzen bereit. OPRF ist ein two-party Protokoll für das Berechnen des Ergebnisses eines PRF's. In einer OPRF Instanz i hat Alice eine Eingabe x_i. Bob lernt einen PRF secret key k_i. Die „obliviousness“ stellt sicher, dass Bob nichts über die Eingabe von Alice lernt. Alice soll allerdings auch nichts über Bob's keys k_i lernen.

Das KKRT batch OPRF Protokoll (Kolesnikov protocol [1]) basiert auf OT-Extension und ist deswegen sehr schnell. Jede OPRF Instanz benötigt nur 4.5_k Bits für die Kommunikation zwischen zwei Parteien und einige weitere Aufrufe für das Anwenden der Hash-Funktion. In einem leistungsstarken Netzwerk können deswegen Millionen OPRF Instanzen in wenigen Sekunden erzeugt werden. Die Keys k jeder Instanz stehen, in einer Relation, können aber in Bezug auf die Sicherheitseigenschaften des Protokolls vernachlässigt werden.

2.3 Oblivious Switching Network

Ein Oblivious Switching Network nimmt eine Eingabe und gibt additive secret shares zurück. Diese bauen auf Switching Networks auf.

2.3.1 Switching Network

Ein SN ist eine Schaltung mit verschiedenen Gattern. Jedes Gatter besitzt primäre ein- und Ausgaben sowie ein programmier Eingabe p. Ein Switching Network besteht aus folgenden Gattern:

- **Multiplexer:** Nimmt k primäre Eingaben und wählt eines davon als Ausgabe, welches durch die programmier Eingabe $p \in [k]$.

- **Permute Switch:** Ist eine Schaltung, welches zwei Eingaben k erhält. Des Weiteren wird dem *Permute Switch* ein boolscher Wert übergeben, der hier ebenfalls als programmier Eingabe bezeichnet wird. Dieser bestimmt dann, ob die Eingaben vertauscht werden sollen. Hier wird von einer Permutation gesprochen, da die Anzahl an gesetzten Bits gleich bleibt und nur deren Reihenfolge geändert wird.

2.3.2 Universal Switching Network

Ein Switching Network $\mathbb{S}$ ist universell einsetzbar, wenn es für jede Funktion der Klasse $\pi : [n_{out}] \to [n_{in}]$ einen Weg gibt, bei dem die programmier Eingabe so gesetzt werden kann, dass das USN $\mathbb{S}$ folgende Abbildung erreicht: $(a_1, ..., 1_{n_{in}}) \to (a_{\pi_{(1)}}, ..., a_{\pi_{(n_{out})}})$

2.3.3 Oblivious Switching Network

Ein OSN ist ein Protokoll welches als Eingabe vom Empfänger ein π erhält, welches die Ausgabemenge auf die Eingabemenge abbildet. Somit ist es mit dieser Funktion möglich, für ein jeweiliges Element das Element aus dem Urbild zu bestimmen. Hierbei ist zu beachten, dass diese Funktion injektiv ist. Als Eingabe vom Sender erhält es einen Vektor $\vec{x}$. Die Ausgabe sind sogenannte **additive secret shares**. Das OSN wird hier wie folgt definiert: $\mathbb{S}^{\pi}(\vec{x})$, wobei $\mathbb{S}$ das Switching Network ist. Hierbei sei erwähnt, dass ein OSN aus mehreren Multiplexern und Permute Switches bestehen kann.

2.3.4 Beispiel

Um ein Einfaches Beispiel zu geben, sei hier ein 1-out-of-4 oblivious transfer. Hierbei besteht das SN aus einem Multiplexer, welcher 3 primäre Eingaben $A_1, ..., A_3$ erhält und eine dazugehörige Ausgabe B.

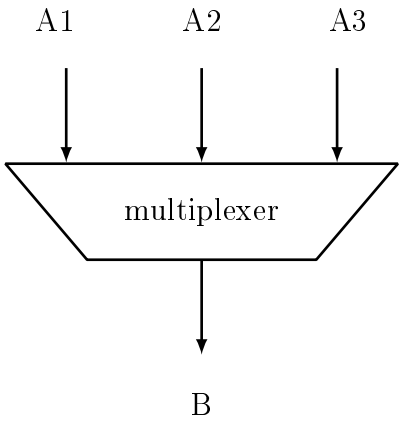


Abbildung 2: 1-out-of-3 Multiplexer

Alice möchte die wissen, ob das Element A_2 mit ihrem übereinstimmt. Sie übermittelt dazu die Programmierung $p \in [3]$. Bob sendet ihr daraufhin $(A_1 \oplus B_b, A_2 \oplus B_b, A_3 \oplus B_b)$. Da Alice die Programmierung gewählt hat, kennt sie die dazugehörige Ausgabemaske B_a für ihre Eingabe. Von daher kennt sie ebenfalls $(B_a \oplus v)$ wobei v ihre Eingabe auf dem Pfad 2 ist, mit welchem sie A_2 vergleichen möchte. Nun überprüft Alice die Eingabe von Bon. Falls

$$\{\exists i \in \{1,2,3\} : (A_i \oplus B_b) \oplus v = B_a\}$$

wahr ist, dann besitzt Bob ebenfalls das Element v.

2.4 Private Equality Tests

Ein Private Equality Test gibt zwei Parteien die Möglichkeit, zwei Eingabestrings mit einander zu prüfen, ohne Preisgabe von Informationen aus diesen.

2.4.1 Beispiel

Alice möchte ihre Eingabe x mit der Eingabe y von auf Gleichheit prüfen $x = y$, ohne ihr x preisgeben zu müssen. Hierzu kann z.B. ein oblivious PRF verwendet werden. Hierbei übernimmt Alice die Rolle des OPRF Empfängers und lernt zu ihrer eigenen Eingabe $PRF(k, x)$, sowie $PRF(k, y)$. Bob hingegen lernt den seed k und die Ausgabe von $PRF(k, y)$.

3 Performance Unterschiede zwischen PSI und PCSI

Wir wissen nun, was man unter den Begriffen PSI und PCSI versteht. Die beiden Anwendungsfälle unterscheiden sich allerdings in der Realität in Bezug auf die Performance. PCSI state-of-art (aktueller Standard) Protokolle sind bis zu 20x langsamer als PSI state-of-art Protokolle. Wie kann diese Lücke überbrückt werden?

Um den Unterschied zu verstehen, betrachten wir zunächst das State-Of-Art Protokoll PSTY19 für das Berechnen von $f(X \oplus Y)$ (PCSI). Am Ende der sog. Preprocessing Phase des Protokolls lernt Alice eine feste Reihenfolge ihrer Items $(x_1, x_2, ..., x_n)$. Alice lernt daraufhin Vektor $\vec{s}$ und Bob lernt Vektor $\vec{t}$, sodass s_i und t_i (Elemente aus den Vektoren) additive secret shares sind. Wenn x_i in der Intersection liegt, dann würde $s_i \oplus t_i = 0$ sein. Andernfalls würde das Ergebnis zufällig aussehen.

Führ das Berechnen einer Funktion über der Intersection müssen alle s_i und t_i miteinander verglichen werden und dann die Funktion darauf angewendet werden $f(comparisons(= X \oplus Y))$. Das benötigt $O(n)$ Vergleiche bei n Elementen. Wir wollen nun einen genaueren Blick auf die Kommunikationskosten dieses Prozessen werfen. In der Abbildung 3 ist zu erkennen, dass der Vergleich von zwei l-Bit Strings im PSTY19 Protokoll , $O(l * k)$ Nachrichten (communications) benötigt. k steht für dabei für den Sicherheitsparameter und l für die Bitlänge.

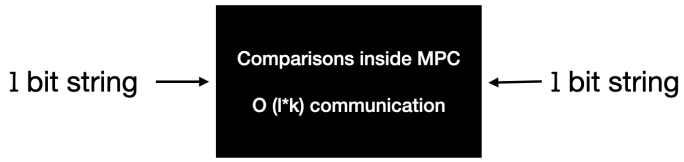


Abbildung 3: String Equality Test in PSTY19

State-Of-Art PSI Protokolle nutzen einen effizienteren Ansatz namens „Special Purpose“ Equality Test für das Vergleichen von Strings. Für das Vergleichen von zwei l-Bit Strings benötigt der Ansatz nur $4.5 * k$ Bits of communication, sodass die Kommunikationskosten unabhängig der Länge der Strings l ist. Allerdings verrät diese Version des Equality Tests jedes Ergebnis eines Vergleiches an eine Partei (immer Alice oder immer Bob). Abbildung 3 zeigt visuell die Idee des „Special Purpose“ Equality Tests.

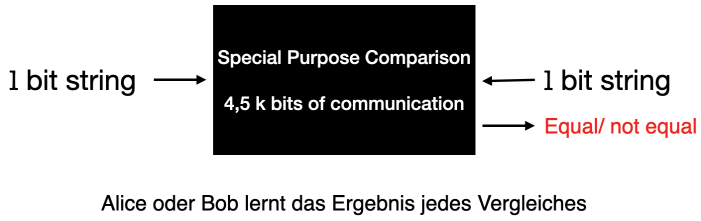


Abbildung 4: String Equality Test in state-of-art Protokollen

4 Ansatz des neuen Protokolls

Die Frage, die sich nach dem Abschnitt 3 stellt ist, können wir „Special Purpose“ Equality Tests in das neue Protokoll einbauen, denn wir haben gesehen, dass das Vergleichen von beliebig langen Strings effizienter ist.

Damit das neue Protokoll P_{PSOFOS}ebenfalls die effizienter Variante in der Preprocessing nutzen kann, wird ein Oblivious Switching Network eingebaut. Das OSN soll die Reihenfolge der Equality Tests permutieren. Das Permutieren benötigt $O(n(log*n)$ Oblivious Transfers. Die Kommunikationskosten sind dadurch $O(n * (log * n) * k)$. Zusammengefasst werden dadurch die Kommunikationskosten von $O(n * l * k)$ auf $O(n * (logn) * k)$(Permutationskosten OSN) $+O(n * k)$(Equality Tests) gesetzt. Die neue Version bringt zwar einen $log * n$ „Overhead“ mit sich, trotzdem ist die Operation in fast allen Fällen effizienter, da der Faktor l nicht mehr vorhanden ist ($log*n < l$ Faktor). P_{PSOFOS} kann allerdings nicht nur für PCSI benutzt werden, sondern es kann jede symmetrische Funktion g(X $\wedge$ Y), die die Kardinalitäten $|X \wedge Y|$ verrät, unterstützen. Dazu zählen folgende Anwendungsfälle.

- Berechnen der PSI
- Berechnen der PSU
- Berechnen der Kardinalitäten der Schnittmenge

- Berechnen der secret-shares der Items in der Schnittmenge
- Realisieren der private ID Funktionalität

5 Permutierte Charakteristik Funktionalität

Um $\mathcal{F}_{pc}$ mittels eines Beispiels darzustellen, werden zwei Parteien definiert. Alice besitzt eine Menge von Daten $A = \{a_1, \ldots, a_n\} \subseteq \{0,1\}^*$. Bob besitzt ebenfalls eine Menge an Daten $B = \{b_1, \ldots, b_n\} \subseteq \{0,1\}^*$. Beide Mengen haben die gleiche Anzahl an Elementen $n = |X| = |Y|$. Alice agiert hier als Empfänger und ist somit diejenige, die den Charakteristik-Vektor $\vec{e}$ lernt. Bob, hingegen lernt die Permutation π. Um dorthin zu gelangen werden folgende Schritte innerhalb der Funktion $\mathcal{F}$ benötigt:

1. Wähle eine zufällige Permutation π über $[n]$. $[n]$ sind alle Indizes $\{1, \ldots, n\}$.
2. Nun definiert die Funktion den Charakteristik-Vektor $\vec{e} \in \{0,1\}^n$, so dass $e_i = 1$ wenn $a_{i\pi_{(i)}} \in B$ sonst $e_i = 0$. Hierbei is zu beachten, dass Bobs Menge B zu diesem Zeitpunkt schon permutiert worden ist. Also die Reihenfolge seiner Elemente wurden geändert. Die Elemente selbst allerdings nicht(Karten werden beim Skart gemischt und nicht geändert).
3. Die Funktion gibt im letzten Schritt den Charakteristik-Vektor an Alice zurück.

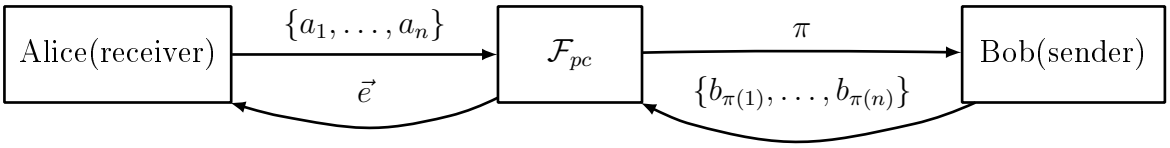


Abbildung 5: $\mathcal{F}_{pc}$

Nun besitzt Alice einen Charakteristik-Vektor $\vec{e}$ welches an jedem Index, ob das Element in Bobs Menge vorhanden ist, wenn der Wert auf 1 gesetzt ist. Die Ausgabe von $\mathcal{F}_{pc}$ kann sowohl für die Berechnung Vereinigungsmenge(PSU) als auch für die Berechnung der Schnittmenge(PSI) verwendet werden. Um die Schnittmenge $S = A \cap B$ zu berechnen, werden alle Elemente aus A, an denen der Index $e_i = 1$ gesetzt ist, zusammengefasst. Die Vereinigungsmenge lässt sich durch Aufrufe von $\mathcal{F}_{ot}$ bestimmen, in dem diese Aufrufe auf alle Indizes i ausgeführt werden, in denen $e_i = 0$ ist. Das Ergebnis der $\mathcal{F}_{ot}$-Aufrufe ist eine Menge $B_{ot} = B\ A$ mit allen Elementen, die nicht in A vorhanden sind. Wird diese Menge nun mit der Menge von Alice Vereinigt, dann ist das Ergebnis die Vereinigungsmenge $V = B_{ot} \cup A = (B\ A) \cup A = B \cup A$ beider Parteien.

Die hier vorgestellte Variante[1, S. 8, Fig. 5] $\mathcal{F}_{pc}$ verwendet für das *Oblivious Switching Network* eine Permutation ohne Multiplexer.

6 Core

Für das Berechnen der Schnittmenge (PSI) oder Vereinigungsmenge (PSU) zweier Sets oder das Anwenden einer Funktion auf der Schnittmenge (PCSI) benötigen wir einen Protokoll Core (P_c), der bei jedem der späteren Funktionalitäten das Basis Protokoll darstellt.

Der Protokoll Core besteht aus drei Funktionalitäten, beginnend mit PSTY19 Preprocessing, dann Oblivious Shuffle und „special purpose“ Equality Tests.

6.1 Preprocessing (PSTY19)

Alice hat ein Set $X = \{x_1, \ldots, x_n\}$ und Bob ein Set $Y = \{y_1, \ldots, y_n\}$. Der erste Schritt der Preprocessing Phase ist, dass Alice ihre Eingaben mit Hilfe von 3 Hashfunktionen $h1, h2$ und $h3$ via Cuckoo Hashing m Bins zuordnet. Jeder Bin hat höchstens ein Item und jedes Item ist nur in einer der drei Position der Hashfunktionen zu finden.

Bob wendet einfaches Hashing auf sein Set an. Dafür benutzt er die selben Hashfunktionen $h1, h2$ und $h3$, die auch Alice benutzt hat. Jedes seiner Items ist 3 Bins zugeordnet. Welchen Bin ein Item zugeordnet wird, bestimmen die Hashfunktionen. Im Gegensatz zu den Bins beim Cuckoo Hashing, kann ein Bin beim einfachen Hashen, mehrere Items besitzen. Für jeden Bin $j \in [m]$, erzeugt Bob anschließend ein zufälliges Geheimnis s_j. Angenommen Alice und Bob besitzen ein gleiches Item in Bin j, dann soll Alice den Wert s_j lernen. In Abbildung 6.1 ist der erste Schritt der Phase visuell dargestellt.

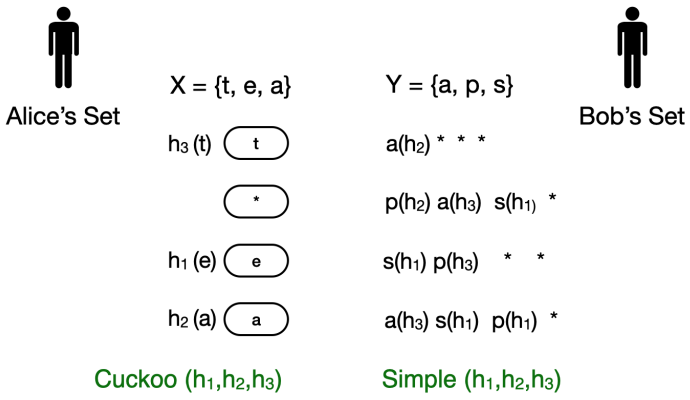


Abbildung 6: Protokoll Core Preprocessing Phase

Im nächsten Schritt benutzen Alice und Bob $\mathcal{F}_{bOPRF}$, um miteinander zu kommunizieren (vgl. Abschnitt 2.2). Wir betrachten das $\mathcal{F}_{bOPRF}$ erst mal als Black-Box. Für jeden Bin j führen Alice und Bob ein $ORPF$ Protokoll aus. Dieser Schritt zusammen mit einem „hint“ , den Bob an Alice sendet, hilft Alice ein $t = \{t_1, \ldots, t_j\}$ zu lernen. Um zu verstehen, wie Alice dieser Schritt hilft, spielen wir das Szenario anhand der Abbildung 6.1 einmal durch. Alice hat im ersten Bin (#1) ein t. Dieser Buchstabe liegt nicht in der Schnittmenge der Eingaben von Alice und Bob. Wenn man nun s_1 von

Bob und t_1 von Alice betrachtet, sind diese additive Shares einer pseudo zufälligen Zahl $s_1/oplust_1 = \$\$$ ($\$\$$ = pseudo zufällige Zahl). Wenn wir den Bin #2 betrachten, sehen wir, dass kein Item dem Bin zugeordnet wurde. Auch hier würde mithilfe von s_2 und t_2 eine $\$\$$ Zahl herauskommen. Pseudo zufällig meint in diesem Zusammenhang, dass die Ausgaben für Alice zu diesem Zeitpunkt willkürlich aussehen. Jetzt betrachten wir den Bin #4. In diesem Bin liegt der Buchstabe a, der in der Schnittmenge von Alice und Bob liegt. Wenn wir s_4 und t_4 betrachten, dann sind diese additive Shares von 0 ($s_1/oplust_1 = 0$).

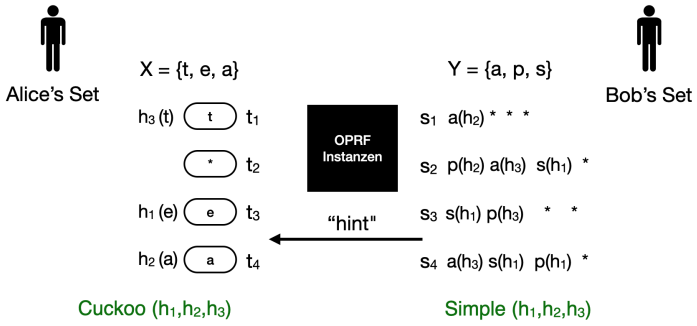


Abbildung 7: Protokoll Core Preprocessing Beispiel

6.2 Oblivious Shuffle

Beim Oblivious Shuffle kommt die Funktionalität $\mathcal{F}_{osn}$ (Oblivious Switching Network) zum Einsatz. Hierbei übernimmt Alice die Rolle des Empfängers und Bob die Rolle des Senders. Alice erstellt eine Funktion $\widetilde{\pi}$: $[n] \to [m]$ so, dass $\widetilde{\pi}(1), \ldots, \widetilde{\pi}(n)$ nicht leere Bins von $\mathcal{A}$ sind. Bob übergibt seinen $\vec{s}$ an das $\mathcal{F}_{osn}$ und Alice die Funktion $\widetilde{\pi}$.

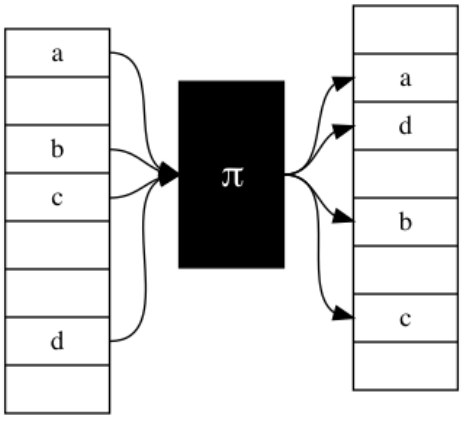


Abbildung 8: Oblivious shuffle

Alice erhält als Ausgabe der Funktionalität $\mathcal{F}_{osn}$ den Vektor $\vec{a}$ und Bob erhält den Vektor $\vec{b}$, so dass $a_i \oplus b_i = s_{\widetilde{\pi}(i)}$ ist. Nun kann Alice lokal den Vektor $\vec{a}'$ als $a'_i = a_i \oplus t_{\widetilde{\pi}(i)}$

berechnen, so dass $s_{\widetilde{\pi}(i)} \oplus t_{\widetilde{\pi}(i)}$ ist. $a'_i = b_i$ ist immer gleich wenn $s_{\widetilde{\pi}(i)} = t_{\widetilde{\pi}(i)}$ und somit sind a'_i und b_i secret shares.

6.3 Equality Tests

Alice und Bob rufen das Protokoll F_{bEQ} auf. Alice ist der Sender und übergibt den Vektor $\vec{a'_i}$. Bob ist der Empfänger und übergibt seinen Vektor $\vec{b}$ und erhält anschließend den Charakteristik-Vektor $\vec{e}$, welcher an der Position i eine 1 besitzt, wenn $x_i = y_i$ und eine 0 wenn nicht.

Um daraus die Schnitt- und/oder Vereinigungsmenge zu berechnen, wird im nächsten Abschnitt beschrieben.

7 Schnitt- und Vereinigungsmenge

In diesem Abschnitt wird die Berechnung der Vereinigungs- und Schnittmenge von Daten zweier Parteien erläutert. Hierzu wird die Funktionalität $\mathcal{F}_{pc}$ eingeführt, welches einen Vektor $\vec{e} \in \{0,1\}$ mit den übereinstimmenden Elementen zurückgibt. *pc* steht hierbei für *private computing*. Aus diesem Vektor wird dann im Nachhinein die Vereinigungs- und Schnittmenge berechnet.

8 PCSI

PCSI (Private Computing on Set Intersection) ist die Bezeichnung auf einer Schnittmenge eine Berechnung durchzuführen. Bei diesem Ansatz des Protokolls werden nach der Berechnung nicht nur das Ergebnis an eine andere Partei gegeben, sondern auch die Kardinalität der Schnittmenge. Die Abbildung 9 zeigt die ideale Funktion $F^{g}_{pcsi+card}$.

Parameters: Size of sets n. Function g.

On input $X \subseteq \{0,1\}^*$ from the sender and $Y \subseteq \{0,1\}^*$ from the receiver:

1. Give $\Big(|X \cap Y|,\ g(X \cap Y)\Big)$ to the receiver.

Abbildung 9: Ideale Funktionalität für das Berechnen der Kardinalität und einer beliebigen Funktion der Schnittmenge

8.1 Kardinalität

In manchen Fällen möchte man keine Funktion g auf die Schnittmenge anwenden, sondern z. B. nur die Kardinalität ausrechnen. In diesem Fall wäre dann die Funktion g leer. Die Kardinalität kann auch durch den beschriebenen Protokoll-Core erhalten werden. Sender und Empfänger führen das Protokoll F_pc auf ihren Eingaben durch. Der Empfänger erzeugt im Protokoll einen Charakteristik-Vektor e und gibt das Hamming Gewichts dieses Vektors zurück. Das Hamming Gewicht eines Strings ist die Anzahl

an Symbolen, die sich von dem Zero-Symbol des Alphabets unterscheiden. Wenn wir das Beispiel auf den String 11101 anwenden, erhält man das Hamming Gewicht, da 4 Einsen im String vorhanden sind und die 1 das Zero-Symbol eines Strings aus Bits ist.

8.2 Kardinalität-Summe

Abbildung 8.2 zeigt ein einfaches Protokoll für die Berechnung der Kardinalität´s Summe (cardinality sum). Mit Hilfe des Charakteristik-Vektors e kann der Empfänger die Summe lernen. Dafür lernt der Empfänger entweder einen zusätzlichen Secret Share 0 oder einen secret share des zugehörigen Wertes dieses Elements. Dann kann der Empfänger lokal zusammenrechnen, indem dieser die secret shares zusammenfügt.

Parameters: Size of sets n. Group $\mathbb{G}$.

Inputs: $X = \{(x_1, v_1), \ldots, (x_n, v_n)\}$ for the sender, where $v_i \in \mathbb{G}$; $Y \subseteq \{0,1\}^*$ for the receiver.

Protocol:

1. Parties invoke $\mathcal{F}_{\mathsf{pc}}$ with inputs $\{x_1, \ldots, x_n\}$ and Y. The Sender obtains a permutation π. Receiver obtains characteristic vector $\vec{e}$.
2. The sender chooses a random vector $(r_1, \ldots, r_n) \leftarrow \mathbb{G}^n$ such that $\sum_i r_i = 0$.
3. Parties invoke n instances of OT via $\mathcal{F}_{\mathsf{ot}}$. The receiver uses $\vec{e}$ as their choice bits. The sender uses input $(r_i, r_i + v_{\pi(i)})$ as input to the ith OT. The receiver gets output $\hat{v}_i$ from the ith OT.
4. The receiver outputs $\left(\sum_i e_i,\ \sum_i \hat{v}_i\right)$.

Abbildung 10: Protokoll für das Berechnen der Kardinalität´s Summe

9 Secret Share Intersection

Nachdem Alice und Bob das Protokoll F_{pc} ausgeführt haben, weiß Bob die Indices von Alice´s Schnittmengen Items. Diese sind allerdings permutiert durch die Funktion pi. Beim Berechnen der PSI/PSU hat Bob deshalb durch OT selektiv Items der Schnittmenge bzw. Differenzmenge gelernt. Jetzt könnte man davon ausgehen, dass Bob einfach die Secret shares der Schnittmengen-Items lernt. Dies ist allerdings ein Ansatz, der nicht funktioniert. Nehmen wir an, dass Alice und Bob jeweils 1 Million Items in ihrem Set haben und 10 davon in beiden Sets liegen (Schnittmenge). Bob könnte OT nutzen, um die secret shares, der 10 Items zu lernen. Jetzt wollen wir in einem 2PC Protokoll die Funktion g auf diese 10 Items anwenden. Alice hat für ihre 1 Million Items ebenso viele Secret Shares erzeugt und kann jetzt nicht explizit die 10 Items der Schnittmenge in g geben, weil sie nicht weiß welche. Bob weiß welche die Richtigen Secret Shares sind, kann dies Alice allerdings nicht mitteilen, da Alice über die permutierte pi Funktion verfügt und damit die gesamte Schnittmenge bestimmen kann. Mit einem weiteren Oblivious Switching Network kann diese Problematik umgangen werden. Alice hat eine geheime Permutation ihrer Items und Bob weiß welche Indices in der Permutation welchen Items in der Schnittmenge entsprechen. Bob wählt eine Injektive Funktion, dessen Bereich genau die Schnittmenge abdeckt. Alice und

Bob benutzen eine OSN, sodass beide Parteien additive Shares nur für die Elemente lernen, auf die durch die Funktion p verwiesen wird.

10 Private-ID

Private-ID ist eine Funktionalität, bei dem bei der Eingabe von zwei Mengen jedes Element ein zufälliger Wert, welcher als ID bezeichnet wird, zugewiesen wird. Alle Elemente aus den Mengen, die gleich sind, erhalten auch die gleiche ID. Im Anschluss erhält jede Partei die IDs der eigenen Elemente zurück sowie die IDs aller Elemente aus den beiden Mengen, welches die Vereinigungsmenge der IDs beider Parteien ist. Nun kann jede Partei für sich weitere Berechnungen auf diesen Mengen ausführen, wie z.B. die Schnittmenge zu bestimmen, indem die Übereinstimmung beider Mengen berechnet wird.

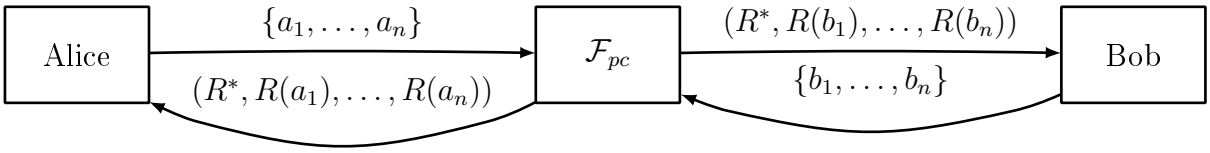


Abbildung 11: $\mathcal{F}_{pc}$

In der wissenschaftlichen Arbeit *Private Set Operations from Oblivious Switching*[1] wird ein weiterer Ansatz beschrieben, welcher hier weiter behandelt wird. In diesem Ansatz wird zur Umsetzung von Private-ID die zuvor vorgestellten Funktionalitäten verwendet. Zum einen wird *OPRF* und zum Anderen *private set union* verwendet.

10.1 Vorgehensweise

Wie zuvor beschrieben, generiert beim *OPRF* die eine Partei Schlüssel k_i, wendet diese mittels *PRF* auf die eigene Menge an, und übergibt die Schlüssel an die zweite Partei. Für das Private-ID wird dieser Vorgang einmal in jede Richtung ausgeführt, sodass Alice die Schlüssel k_A generiert und Bob die Schlüssel k_B. Sie übergeben diese dann an die jeweils andere Partei. Alice besitzt nun k_A, k_B und ihre Menge an Eingaben A. Zudem kennt sie durch *OPRF* noch die die Ausgaben von $PRF(k_A, a_i)$ für jedes Ihrer Elemente $a_i \in A$ Bob besitzt ebenfalls die Schlüssel k_A und k_B. Zudem kennt er seine eigene Menge B sowie $PRF(k_b, b_i)$ für jedes seiner Elemente $b_i \in B$. Nun fehlt nurnoch die zufällige ID der Elemente x welches bei Alice a und bei Bob b ist. Diese wird in der Arbeit wie folgt definiert:

$$R(x) \stackrel{def}{=} PRF(k_A, x) \oplus PRF(k_B, x)$$

Da beide Parteien die Schlüssel k_A und k_B besitzen, können beide $R(x)$ für deren eigenen Elemente berechnen. Da Alice für sich bereits $PRF(k_A, a)$ für jedes Element

a berechnet hat, fehlt ihr nur noch die Berechnung von $PRF(k_B, a)$ für jedes ihrer Elemente a. Im Anschluss kann sie mit XOR $R(a)$ für jedes ihrer Elemente berechnen. Das Gleiche gilt für Bob. Nun wird private set union auf die Elemente $R(a)$ für jedes $a \in A$ und $R(b)$ für jedes Element aus $b \in B$ angewandt. Das Ergebnis ist die Vereinigung aller IDs. Auf diesem können nun wie am Anfang der Sektion beschrieben fortgefahren werden.

11 Komplexitätsvergleich

11.1 PCSI

Das aktuell führende Protokoll für PCSI Protokolle ist das PSTY19, dessen ersten Schritte auch im Core-Protokoll zu finden sind (siehe Abschnitt 6.1). Wir vergleichen deswegen die Protokollschritte dieses Protokolls mit dem PSTY19 Standard nach dem Schritt des Preprocessing´s, da beide in diesem Teil identisch sind.

In PSTY19 folgenden $1,27 * n$ private equality tests in einem MPC. n steht für die Anzahl an Items. Die Kosten eines solches PEqT´s sind $2 * l * k$. Die Gesamtkosten sind damit:

$$2,54 * l * k * n$$

In diesem Protokoll folgt dem Preprocessing ein Oblivious Switching Network mit

$$1,27 * n * logn$$

Knoten. Jeder Knoten benötigt OT auf Strings der Länge $2 * l$. Die Kosten für ein OT sind $k * 4 * l$ Bits. Die Gesamtkosten sind damit:

$$1,27 * (n * log * n)(k + 4l)$$

Gegenübergestellt und mit realen Werten für $l = 60$ und $k = 128$ eingesetzt ergibt sich dann folgende Rechnungen:

$$\begin{aligned} 1,27 * (n * log * n)(k + 4l) = & \\ &= 1,27 * (n * logn)(128 + 4 * 60) \\ &= 1,27 * (n * logn)(k + 4l) \\ &= 1,27 * 368 * (n * logn) \\ &= 467 * logn \end{aligned}$$

&

$$\begin{aligned} 2,54 * l * k * n = & \\ &= 2,54 * 60 * 128 * n \\ &\approx 19500 * n \end{aligned}$$

Wenn wir nun n aus beiden Ergebnissen entfernen, denn n wird beim Ersetzen identisch sein, dann stellen wir fest, dass für ein beliebiges n, die Kosten des neuen Protokolls geringer sind (bei $n = 2^{41}$ ist es nicht mehr der Fall, ist aber sehr unrealistisch).

11.2 PSU

Um die Performance der Bestimmung der Vereinigungsmenge zweier Mengen mittels des **PSTY19**-Protokolls wird dieses hier in Kontext gestellt mit dem Protokoll von *Kolesnikov*. Hierbei soll gezeigt werden, wie groß der Vorteil des Protokolls **PSTY19** ist.

11.2.1 Kolesnikov

Es wird sich an die Definition von Bins erinnert. In einem Bin können mehrere Items abgelegt werden. Beim Cuckoo-Hashing hingegen wird nur in jedem Bin ein Item abgelegt. Beim *Kolesnikov* hingegen werden mehrere Items abgelegt und noch ein zusätzliches Padding angefügt, um die Anzahl der Elemente zu verschleiern. In diesem Protokoll werden n Items in $m = O(n/logn)$ Bins gelegt. Hierbei ist die erwartete Anzahl an Elementen in jedem bin n/m, wobei im worst-case dieser um einen konstanten Faktor größer ist. Wie zuvor besprochen werden Items angehängt. Hier wird auch von Padding gesprochen. Hierbei wird um die Anzahl an Elementen zu verschleiern, Items bis zum worst-case angehängt. Dieser Faktor wird als c bezeichnet. Somit liegt die Anzahl an Elementen in jedem Bin nicht bei n/m sondern bei cn/m. Um die Vereinigungsmenge bestimmen zu können, muss für jeden Bin eine lineare Anzahl an OPRFs und OTs ausühren und zur Potenz 2 Kommunikationskosten zur Anzahl an Items. D.h. für β Items in einer Bin entstehen Kommunikationskosten von $\beta^2\sigma$, wobei $\sigma = \lambda + 2logn$ ist und λ der Sicherheitsparameter ist. Zusammengefasst heißt es, dass die Kommunikationskosten dieses Protokolls wie folgt ist:

$$cn \cdot bOPRF + cn \cdot OT + (c^2 nlogn)\sigma$$

Hierbei sind $bOPRF$ und OT die jeweiligen Kommunikationskosten der Unterprotokolle.

11.2.2 PSTY19

Wird nun das vorgeschlagene **PSTY19**-Protokoll in Kontrast zum Protokoll von *Kolesnikov* gestellt, sind einige Unterschiede zu erkennen. Beim **PSTY19** wird für das Senden von einem Polynom des dritten Grades $1,27n$ OPRFs für das Preprocessing benötigt. Des Weiteren ca. $1,27nlogn$ OTs für das switching Netzwerk sowie n zusätzliche OTs für die Bestimmung der Vereinigungsmenge($+n$ innerhalb der Klammer). Hieraus ergibt sich folgende Gleichung für die Kommunikationskosten:

$$1,27n \cdot bOPRF + 3n\sigma + (1,27nlogn + n) \cdot OT$$

Wird dieser nun mit dem Vorherigen verglichen, wird ersichtlich, dass c, welches der worst-case Faktor ist, auf eine Konstante $1,27$ gesetzt. Dies liegt an dem Preprocessing. Des Weiteren wird in der Klammer c^2n zu $1,27n$, da durch das verwendete Beneš-Netzwerk die obere Grenze die Konstante 1 ist. Der signifikante Unterschied zwischen den Protokollen ist der Ausdruck, welcher $O(nlogn)$ beinhaltet. Das Protokoll

PSTY19 ist dem anderem überlegen, solange die folgende Bedingung erfüllt ist:

$$1,27OT < c^2\sigma$$

12 Performancevergleich

Es wird ein Testaufbau zwischen den bestehenden Implementierungen und dem Protokoll **PSTY19** aufgezeigt. Dieser Vergleich soll hervorheben, unter welchen Bedingungen das Protokoll **PSTY19** den anderen Implementierungen überlegen ist. Die Testumgebung ist wie folgt:

LAN	10 Gbps
LAN Latenz	0,02 ms
WAN	50 Mbps
WAN Latenz	80 ms
Eingabegrößen	$n = \{2^{12}, 2^{16}, 2^{20}\}$
Iterationen	5

Tabelle 1: Testumgebung

12.1 Kardinalität

Abbildungen figures 12 to 14 zeigen den Vergleich von PSTY19 und diesem Protokoll in Bezug auf Berechnen der Kardinalitäten einer Schnittmenge (Cardinality of intersection). In Abbildungen 12 13 werden die Umgebungen LAN und WAN gegenüber gestellt und in Abbildung 14 werden die Kommunikationskosten in Megabyte verglichen. PSTY19 benötigt in der LAN-Umgebung bei beliebig großen Eingabemengen weniger Zeit zum Berechnen als P_{PSOFOS}. Allerdings können wir erkennen, dass P_{PSOFOS} in einer WAN-Umgebung schneller berechnet als PSTY19. Darüber hinaus besitzt PSTY19 bei jeder Größe von Eingabedaten höhere Kommunikationskosten (siehe Abbildung 14) und ist somit ineffizienter.

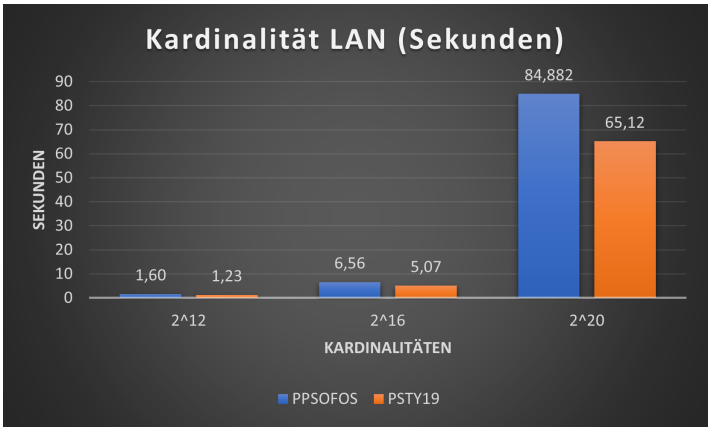


Abbildung 12: Vergleich PSTY19 gegen P_{PSOFOS} in LAN Umgebung für Kardinalität

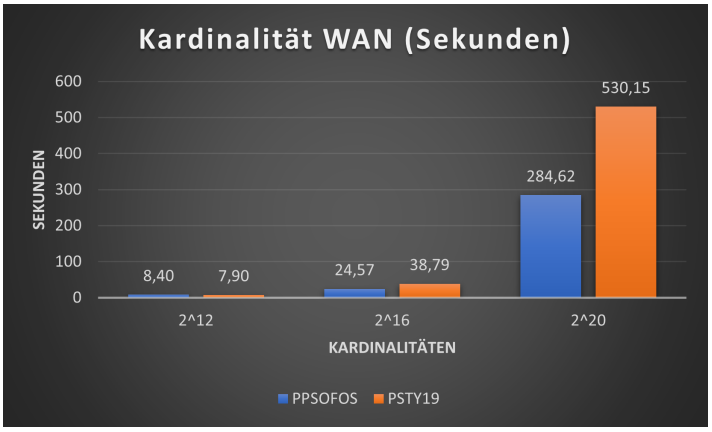


Abbildung 13: Vergleich PSTY19 gegen P_{PSOFOS} in WAN Umgebung für Kardinalität

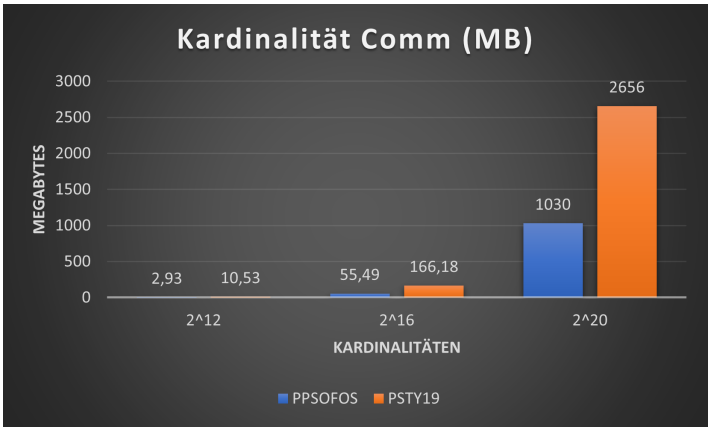


Abbildung 14: Vergleich der Kommunikationskosten von PSTY19 gegen P_{PSOFOS} für Kardinalitätsberechnung

12.1.1 PSU

Wird die Laufzeit von P_{PSOFOS} mit dem Protokoll von *Kolesnikov* [KKRTW19] verglichen, ist zu sehen, dass bei geringen Datenmengen die Laufzeit etwas höher ist als beim Protokoll von *Kolesnikov*. Hingegen bei größeren Datenmengen wie z. B. bei 2^{20} ist die Laufzeit im LAN um mehr als das doppelte geringer.

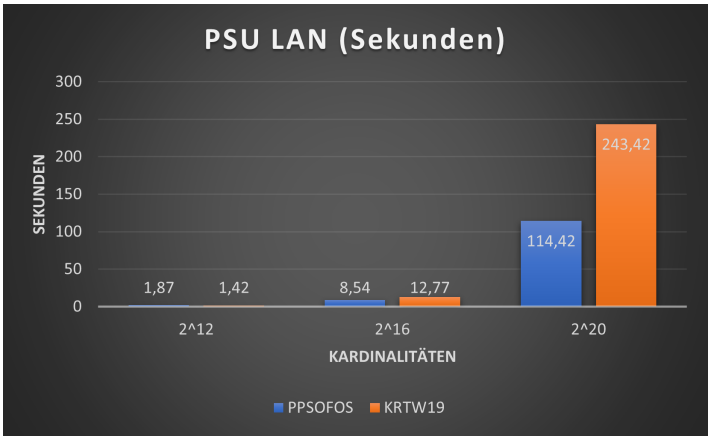


Abbildung 15: Vergleich KKRTW19 gegen P_{PSOFOS} in LAN Umgebung für PSU

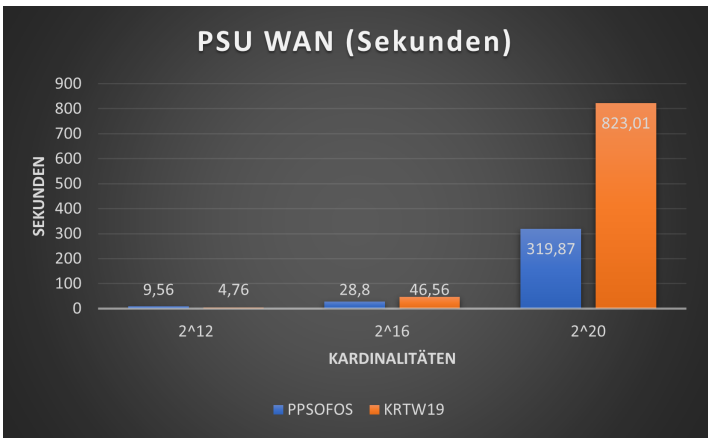


Abbildung 16: Vergleich KKRTW19 gegen P_{PSOFOS} in WAN Umgebung für PSU

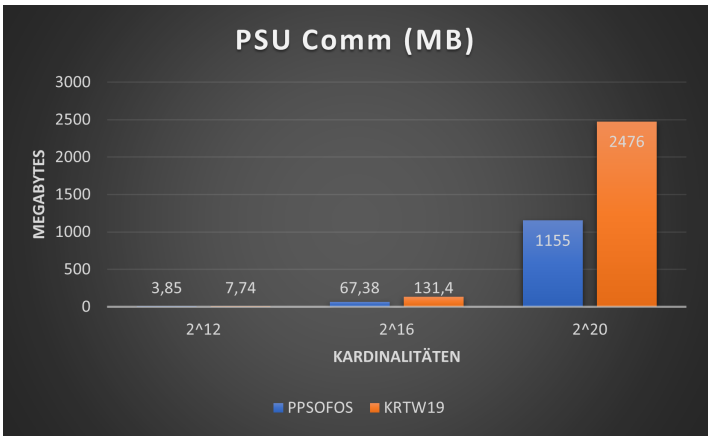


Abbildung 17: Vergleich der Kommunikationskosten von KKRTW19 gegen P_{PSOFOS} für PSU

12.1.2 Private-ID

Beim vergleich zu $BKM^{+}20$ sei zu erwähnen, dass das Subprotokoll aus **PSTY19** ein symmetrisches Verschlüsselungsverfahren nutzt und dass das Protokoll $BKM^{+}20$ ein asymmetrisches Verfahren einsetzt. Das Subprotokoll aus **PSTY19** zieht sein Vorteil aus dieser Differenz, da ein symmetrisches Verschlüsselungsverfahren bei größeren Datenmengen eine höhere Performance und somit eine geringere Laufzeit aufweist. Bei diesem OT-basierten Verfahren mit symmetrischer Verschlüsselung ist ebenfalls zu sehen, dass die Kommunikationskosten deutlich höher sind.

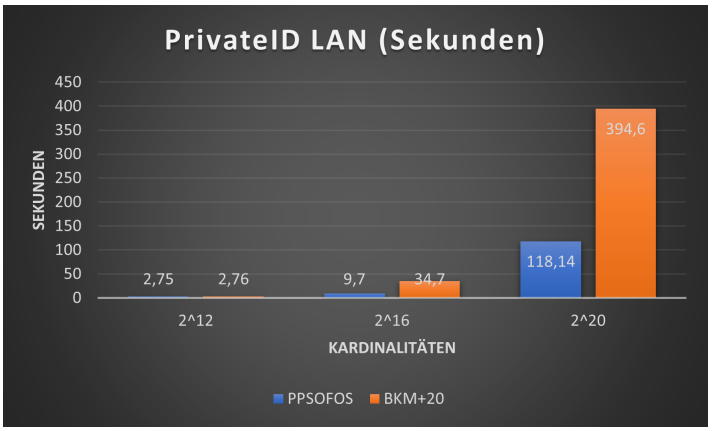


Abbildung 18: Vergleich $BKM^{+}20$ gegen P_{PSOFOS} in LAN Umgebung für PrivateID

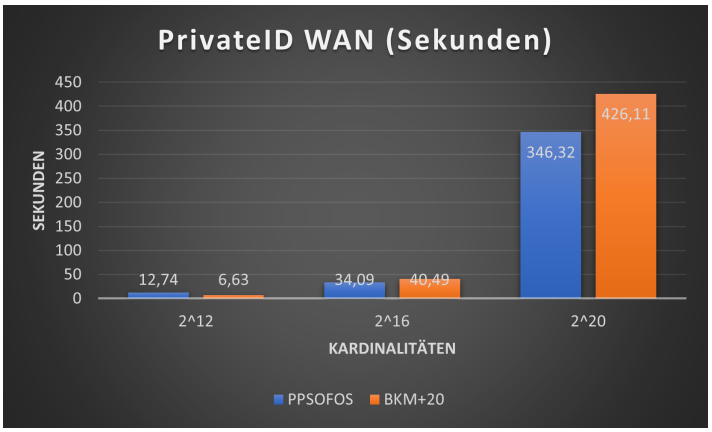


Abbildung 19: Vergleich $BKM^{+}20$ gegen P_{PSOFOS} in WAN Umgebung für PrivateID

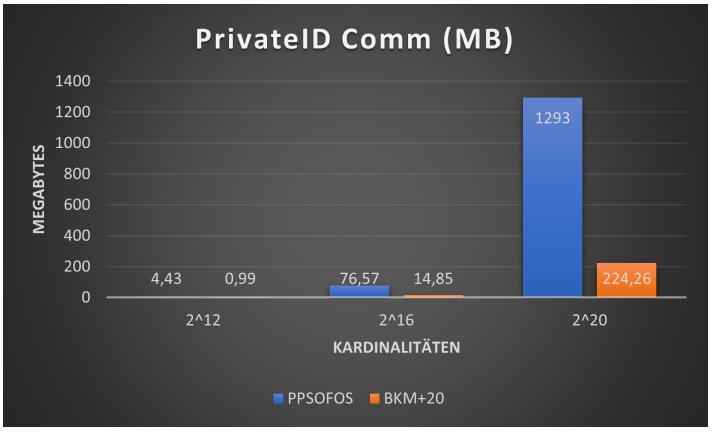
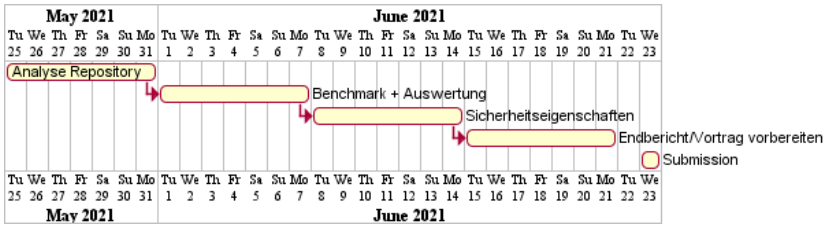


Abbildung 20: Vergleich der Kommunikationskosten für $BKM^{+}20$ gegen P_{PSOFOS} für PrivateID

13 Ausblick

Die weitere Arbeitsplanung haben wir in 4 Phasen aufgeteilt, denn uns stehen 4 Wochen zu Verfügung. Da wir im zweiten Teil des Semesters praktische Ansätze dieses Protokolls untersuchen sollen, haben wir in den letzten Wochen bereits immer mal wieder Researching zum Thema **Private Set Operations from Oblivious Swichting** betrieben. Dabei sind wir auf ein Repository gestoßen, dass von 2 der 6 an dem Paper Arbeitenden gepflegt wurde [`https://github.com/osu-crypto/PSI-analytics`]. Außerdem haben wir durch Links auf dem Paper Zugang zu weiteren Repositories zum Thema. In Abbildung 13 wird unsere Arbeitsplanung als Gantt Diagramm dargestellt.



13.1 Phase 1 (25.05-01.06)

In der ersten Phase soll das Verständnis der wissenschaftlichen Arbeit mittels den zur Verfügung gestellten Repositories vertieft werden. Hierzu sollen die ebenfalls zur Verfügung gestellten Tests ausgeführt und ausgewertet werden. Es geht darum, sich mit dem Programmcode auseinanderzusetzen und Abstraktionen der einzelnen Funktionalitäten, wenn möglich zu verstehen.

13.2 Phase 2 (01.06 - 08.06)

Es sollen unter dem Protokoll Benchmarks mit unterschiedlichen Größenordnungen ausgeführt, eine Auswertung erstellt und hierzu noch Grafiken ausgearbeitet werden.

In der aktuellen Arbeit, die wir als Recherche bekommen haben, sind immer nur drei Größenordnungen getestet worden und wir wollen untersuchen, wie sich das Protokoll bei beliebigen Parametern verhält.

13.3 Phase 3 (08.06 - 15.06)

In der dritten Phase sollen die Sicherheitseigenschaften des Protokolls bestimmt werden. In der ersten Version der Abgabe haben wir diese nicht beachten müssen. Allerdings ist auch dieser Aspekt für die Endabgabe relevant. Es soll betrachtet werden, unter welchen Eigenschaften das Protokoll gegen Semi-Honest Adversaries sicher ist und in welchem Maße dieses Protokoll Änderungen unterzogen werden muss, um die Sicherheit gegen Malicious Adversaries zu gewährleisten.

13.4 Phase 4 (15.06 - 22.06)

In der letzten Phase soll die Quintessence der Arbeit in Zusammenhang mit den erlangten Erfahrungen aus den vorherigen Phasen zusammengefasst und für die restlichen Personen aus der Veranstaltung mittels grafischen Darstellungen, wiedergegeben werden.

References

[1] Gayathri Garimella et al. *Private Set Operations from Oblivious Switching*. Cryptology ePrint Archive, Report 2021/243. `https://eprint.iacr.org/2021/243`. 2021.