A-SIT

Zentrum für sichere Informationstechnologie - Austria

Evaluation von “Secure-Multi-Party-Computation”-Frameworks im Web-Browser

Evaluation von "Secure-Multi-Party-Computation"-Frameworks im Web-Browser

Autor:
Karl W. Koch
Tel: +43 316 873 - 5517
Mail: karl.koch@iaik.tugraz.at

Datum: 31.03.2024

Kurzfassung:

„Secure Multi-Party Computation" (MPC) ist ein mittlerweile praktikabler privatsphären-bewahrender kryptographischer Baustein. Mithilfe von MPC können zahlreiche Daten-Analysen so durchgeführt werden, dass die Eingabe-Daten der Teilnehmer geheim bleiben, und sozusagen die Privatsphäre gewahrt bleibt. So kann, z.B., eine Umfrage in einer Besprechung/einem Vortrag durchgeführt werden, wo nur das Ergebnis bekannt wird und kryptografisch – auf Protokollebene – sichergestellt ist, dass man praktisch keine identifizierbaren Rückschlüsse auf die jeweiligen Teilnehmer ziehen kann. Und idealerweise nehmen die Teilnehmer aktiv an der Berechnung teil.

Einer der nächsten Schritte, um MPC für eine Vielzahl von End-Anwendern zu ermöglichen – wie im genannten Beispiel - ist die Ausführung der MPC-Frameworks auf den jeweiligen End-Geräten. Vor allem weil die Computer-Nutzungsumgebung sehr dynamisch geworden ist, und der Trend weiter in diese Richtung geht - Computer/SmartPhone/Tablet/SmartWatch/SmartHome/etc. - wird die Ausführung im Web-Browser immer relevanter für entsprechende Anwendungen.

Deshalb wurden in diesem Projekt (1) aktuelle Ansätze, um MPC im Web-Browser auszuführen untersucht, und (2) anschließend zwei konkrete MPC-Frameworks (JIFF und MPyC-Web) näher beleuchtet und, basierend auf ihrer Praxistauglichkeit MPC im Browser auszuführen, verglichen.

Article I. Inhaltsverzeichnis

0. Abkürzungsverzeichnis

InProd	„Inner Product" / „Dot Product" (Inneres Produkt bzw. Skalarprodukt zweier Vektoren)
IoT	„Internet of Things" (Internet der Dinge)
JIFF	*JavaScript library for web-based applications that employ secure multi-party computation*
ML	„Machine Learning" (maschinelles Lernen)
MPC	„Secure Multi-Party Computation" (sichere Mehrparteien-Berechnung)
MPC-Motor	*bezeichnet ein MPC-Framework*
MPyC	Multi-Party Computation in Python
MPyC-Web	*die Web-Version von MPyC*
Wasm	WebAssembly

1. Einleitung

„Secure Multi-Party Computation" (sichere Mehrparteien-Berechnung / MPC) ist ein mittlerweile praktikabler privatsphären-bewahrender kryptografischer Baustein. Mithilfe von MPC können zahlreiche Daten-Analysen bzw. Berechnungen im Allgemeinen so durchgeführt werden, dass die Eingabe-Daten der Teilnehmer geheim bleiben, und sozusagen die Privatsphäre gewahrt bleibt. So kann, **z.B., eine Umfrage/ein Wahldurchgang („Voting") spontan in Echtzeit in einer Besprechung/Sitzung/einem Vortrag** durchgeführt werden, wo **nur das Ergebnis bekannt wird** und kryptografisch/mathematisch – auf Protokollebene – sichergestellt ist, dass man praktisch keine identifizierbaren Rückschlüsse auf die jeweiligen Teilnehmer ziehen kann. Das Teilnehmen an einer solchen Umfrage/einem solchen Wahldurchgang **mittels MPC** hat den Vorteil, dass einerseits **niemand sonst die Eingabe der jeweiligen Person lernt** – was tendenziell zu offeneren bzw. ehrlicheren Antworten führen kann - und man andererseits kryptografisch/mathematisch sicherstellen kann, dass die **teilnehmenden Personen korrekte Daten eingegeben** bzw. abgestimmt haben. Und idealerweise, um die Sicherheit während der Berechnung zu maximieren, nehmen die jeweiligen Parteien, welche Daten zur Verfügung stellen, aktiv an der Berechnung teil (= sind Teilnehmer).

MPC-Einstiegshürden. Um aktiv an einer gemeinsamen MPC-Berechnung teilzunehmen oder auch nur um zu experimentieren, muss man das entsprechende MPC-Framework (bzw. MPC-Motor im Folgenden) „aufsetzen" bzw. einrichten. Zum Beispiel, alle benötigten Software-Pakete bzw. -Bibliotheken zu installieren, das Netzwerk einzurichten und schlussendlich das MPC-Programm für jede/n Teilnehmer/in – in der jeweiligen Systemumgebung – zur erfolgreichen Ausführung zu bringen. Obwohl viele MPC-Motoren Docker-Container mit leicht zugänglichen Tutorial-Programmen bereitstellen, ist es immer noch eine anfängliche Hürde, alles für die jeweiligen Experimente und Berechnungen erfolgreich „aufzusetzen" bzw. zu aktivieren. Diese Einstiegs-Hürde ist vor allem hoch, wenn man nicht vertraut mit MPC-Motoren ist und/oder die jeweiligen Teilnehmer keine Technik-Experten bzw. nicht-technik-affin sind.

Einer der nächsten Schritte, um MPC nun für eine Vielzahl von End-Anwendern bzw. potentiellen Teilnehmern zu ermöglichen – wie im oben genannten Beispiel – ist das einfache Ausführen der MPC-Motoren. Um somit die potentielle Einstiegshürde so klein wie möglich zu gestalten, und demnach idealerweise nichts installieren und selbst „aufsetzen" bzw. einrichten zu müssen. Darüber hinaus muss diese Einstiegshürde minimal für alle potentiellen End-Geräte der Teilnehmer gehalten werden. Vor allem, weil die Computer-Nutzungsumgebung sehr dynamisch geworden ist, und der Trend weiter in diese Richtung geht. Denn Personen besitzen immer häufiger eine Vielzahl an technischen Geräten, welche als End-Gerät für MPC in Frage kommen könnten: Computer / Laptop / SmartPhone / Tablet / SmartWatch / SmartHome-Geräte / etc. Beispielsweise kann es sein, dass manche Personen mit dem Computer/Laptop an der/dem oben genannten Umfrage/Wahldurchgang teilnehmen, jedoch andere Personen nur mit dem SmartPhone oder Tablet teilnehmen können.

Warum MPC im Web-Browser? Eine Möglichkeit, um MPC auf einer Vielzahl von End-Geräten zu ermöglichen, ohne dass Teilnehmer die MPC-Motoren selbst einrichten müssen, sind plattform-spezifische Anwendungen (Apps). D.h. es würde für die Teilnehmer Apps für Android / iOS / iPad / Linux / Windows / macOS / etc. geben. In diesem Falle muss aber für jede Plattform eine App in der jeweiligen Programmiersprache erstellt werden. Und obwohl es inzwischen einige Ansätze gibt, um eine App in nur einer Programmiersprache bzw. einem Framework zu erstellen, sodass die App dann auf einigen End-Geräten verwendet werden kann, ist es trotzdem ein beachtlicher Mehraufwand so eine App für alle potentiellen End-Geräte zu erstellen und auch zu warten.

Was nahezu alle End-Geräte in diesem Zusammenhang verbindet – bzw. was zusammengefasst die meisten Personen in den genannten Szenarien anspricht – ist der Zugang zu einem Web-Browser. Gegeben, dass die Web-Browser-Version von, z.B., Safari, Firefox, Edge oder Chrome, bei den jeweiligen Personen aktuell ist, ist grundsätzlich eine homogene Ausführungsumgebung zwischen, z.B., Laptop, SmartPhone und Tablet gegeben. Durch diese über zahlreiche End-Geräte-übergreifende homogene Ausführungsumgebung, wird die Nutzung des Web-Browsers für zahlreiche Anwendungen stetig interessanter und relevanter.

Besonders im Bereich von MPC, hat der Web-Browser im Vergleich zu System- oder mobilen Anwendungen den Vorteil, dass die Teilnehmer nichts installieren müssen; sie brauchen „lediglich" einen kompatiblen Web-Browser. Und da keine Installation erforderlich ist, ist das „Aufsetzen" bzw. Einrichten für die Teilnehmer um einiges reibungsloser. Diese reibungslose Einrichtung macht es einfacher, den Mehrwert von MPC – z.B. in Bezug auf die Privatsphäre - für jede Person zugänglich zu machen. Und somit ermöglicht MPC im Web-Browser sogar Nicht-Experten bzw. nicht-technik-affinen Teilnehmern via MPC in Echtzeit ihre Daten einzugeben bzw. ihre Stimme abzugeben; wie z.B. bei den oben genannten Szenarien.

Weiters berichteten, z.B., bereits im Jahr 2018 Barak et al. (Barak, Hirt, Koskas, & Lindell, 2018) die gleiche Relevanz MPC einer Vielzahl an Personen zur Verfügung zu stellen. Sozusagen, dass MPC ein „Standardprodukt" wird; wie der sichere Netzwerk-Verkehr im Internet über TLS („Transport Layer Security"; z.B. in den Web-Browsern Firefox und Safari über das Schloss-Symbol in der Adressleiste gekennzeichnet). Deshalb haben Barak et al. in ihrem Experiment mehrere Möglichkeiten zur Verfügung gestellt, sich aktiv an einer MPC-Berechnung zu beteiligen; und darunter auch über einen Web-Browser. Weiters erläutern Barak et al. ebenfalls die Bedeutung und den Vorteil des Web-Browsers für Teilnehmer:

„A far more compelling way of carrying out the above study would be to have each company run its own copy of the MPC. However, as described by [20], none of these companies would install software and would only use a browser interface." (*„Eine vielversprechendere Lösung, die genannte Studie durchzuführen, wäre wenn jede Organisation selbst bei der MPC-Berechnung aktiv teilnimmt. Jedoch, wie in [20] geschildert, möchte keine dieser Organisationen dedizierte Software installieren, und würde nur über einen Browser teilnehmen."*) (Barak, Hirt, Koskas, & Lindell, 2018)

und

„As the browser is becoming the new OS for end users, additional features and capabilities are constantly added, such as access to device (orientation, location), new Web Cryptography API, and WebAssembly. These open the possibility of achieving high performance MPC directly in the browser." (*„Da der Browser das neue Betriebssystem*

(OS) für End-Anwender wird, werden zusätzliche Funktionalitäten und Kapazitäten im Browser hinzugefügt (wie z.B. Zugriff auf Geräteinformationen bzw. -Sensoren (wie Ortung), eine neue Web-Kryptographie-API und WebAssembly). Diese Erweiterungen ermöglichen es, direkt im Browser hoch-performante MPC-Berechnungen durchzuführen"). (Barak, Hirt, Koskas, & Lindell, 2018)

Ziele & Ausblick. Deshalb werden in diesem Projekt zuerst (Potentielle) Anwendungsfälle von MPC im Web-Browser aufgezählt und anschließend beschrieben wie man diese MPC-Web-Frameworks im Web-Browser ausführen kann. Im Zuge dessen werden zwei konkrete MPC-Frameworks bzw. -Motoren näher beleuchtet, und basierend auf ihrer Praxistauglichkeit verglichen: JIFF und MPyC-Web. Abschließend werden im Rahmen einer Conclusio & Weiterführende Arbeiten, die gewonnenen Erkenntnisse kurz zusammengefasst und potentiell-interessante weiterführende Richtungen aufgezeigt.

2. (Potentielle) Anwendungsfälle von MPC im Web-Browser

In diesem Abschnitt werden weitere potentiell interessante und relevante Anwendungsfälle, um MPC im Web-Browser auszuführen, aufgezählt. Einerseits wurde im vorigen Abschnitt bereits ein Beispiel mit aktiven Teilnehmern in einer Besprechung/Vortrag/Sitzung genannt. Daraus abgeleitet kann MPC im Web Browser – durch die Einfachheit - im Allgemeinen zahlreiche Anwendungsfälle begünstigen bzw. ermöglichen. Weiters ist MPC im Web-Browser im Speziellen, z.B., auch bei hybriden Anwendungen (dedizierter Server und relativ-einfach zu-verwendete Web-Oberfläche) und „IoT"-Szenarien interessant.

Universell: Einfacher MPC-Zugang. Theoretisch überall wo eine Person oder Organisation aktiv bei einer MPC-Berechnung teilnehmen will, kann die Ausführung über einen Web-Browser hilfreich sein. Denn wie im A-SIT-Bericht „Evaluierung von MPC's Stand der Technik" von Anfang 2024 (Koch, 2024) beschrieben, wird MPC immer praxistauglicher, und im Zuge dessen auch häufiger eingesetzt. Wird nun die MPC-Einstiegshürde reduziert bzw. auf ein Minimum gebracht, und sozusagen die Ausführung für eine Vielzahl an potentiellen Teilnehmern ermöglicht, können dadurch einerseits mehr Personen bzw. Organisationen aktiv teilnehmen, und andererseits kann es dazu führen, dass zusätzliche Anwendungsfälle durch diese Einfachheit ermöglicht werden.

Hybride MPC-Motoren. So wie es bei den Autos den hybriden Antrieb mit Strom und Diesel/Benzin gibt, so kann man sich einen „hybriden MPC-Motoren"-Anwendungsfall so vorstellen, dass ein Teilnehmer über einen nativen MPC-Motor bzw. Anwendung teilnimmt (z.B. dedizierter Server in einer Linux-Umgebung via MPyC (Schoenmakers, MPyC: Multiparty Computation in Python, 2024)), und ein anderer Teilnehmer über einen Web-MPC-Motor im Web-Browser (z.B. MPyC-Web (Nikolov & Schoenmakers, 2024)) teilnimmt. Ein praktisch-relevantes Szenario kann hierbei die Inferenz bei maschinellem Lernen (ML) darstellen. Hierbei stellt der Teilnehmer mit dem nativen MPC-Motor ein ML-Modell bereit, und der Teilnehmer mit dem Web-MPC-Motor gibt Daten für die ML-Auswertung ein, und hat lediglich Zugang zu einem Web-Browser für die MPC-Berechnung. Der Mehrwert bei diesem asymmetrischen bzw. hybriden Anwendungsfall, ist einerseits die Zugänglichkeit durch den Web-Browser und andererseits die grundsätzlich höhere Rechen-Kapazität beim nativen MPC-Motor. Z.B. haben Goyal, Liu-Zhang, Ostrovsky 2023 diesen Teil-Bereich von MPC untersucht und als „Asymmetric MPC" (asymmetrisches MPC) bezeichnet (Goyal, Liu-Zhang, & Ostrovsky, 2023). Und wie bei ML-Auswertungen via MPC üblich, bleiben die Eingabedaten des Auswertungs-Teilnehmers geheim, und auch die ML-Modell-Parameter des Servers bleiben geheim.

Darüber hinaus können Anwendungsfälle mit mehreren Teilnehmern und hybriden MPC-Motoren auch relevant sein, um den Teilnehmern die Option zu bieten, zwischen MPC-Motoren in einer nativen Umgebung und im Web-Browser zu wählen. Somit könnten, z.B., manche Teilnehmer des oben genannten Beispiels – in der Besprechung/Präsentation/Sitzung – ihre Daten über einen nativen MPC-Motor in Linux eingeben, und die anderen Teilnehmer über einen Web-MPC-Motor auf dem SmartPhone.

IoT et al. Darüber hinaus kann es auch interessant und relevant sein, weitere – über den Web-Browser hinausgehende – Plattformen zu unterstützen. Wenn der MPC-Motor via WebAssembly (Wasm) im Web-Browser

ausgeführt wird, ist es grundsätzlich auch möglich den MPC-Motor auf verschiedensten „Internet-der-Dinge" (IoT / „Internet of Things")-Geräten auszuführen[1]. Dann wäre es beispielsweise möglich, dass mehrere IoT-Geräte gemeinsam eine Funktion via MPC berechnen, wobei das jeweilige Gerät nur das Resultat lernt. Z.B., wenn das Ampel-System bei mehreren Kreuzungen die Verkehrs-Auslastung in einer MPC-Berechnung bestimmen würde.

3. MPC-Web-Frameworks bzw. MPC-Web-Motoren

In diesem Abschnitt werden zuerst generell die Ansätze von MPC-Frameworks bzw. -Motoren im Web erläutert, und anschließend zwei MPC-Web-Motoren - basierend auf ihrer Praxistauglichkeit – näher betrachtet: JIFF und die Web-Version von MPyC (MPyC-Web).

Stand der Technik von MPC-Motoren. Basierend auf dem A-SIT-Bericht „Evaluierung von MPC's Stand der Technik" von 2024 (Koch, 2024), gibt es grundsätzlich zwei Klassifikationen bei MPC-Motoren: (1) wie viele Parteien aktiv bei der MPC-Berechnung teilnehmen können (=Anzahl Teilnehmer), und (2) für welche Art von MPC-Berechnungen der jeweilige MPC-Motor spezialisiert wurde (= z.B. generisch oder für privatsphären-bewahrendes ML). In dem vorhin genannten A-SIT-Bericht wurden auch die zurzeit aktuellsten bzw. relevantesten MPC-Motoren aufgezählt. Darin gab es nur einen MPC-Motor der direkt in einem Web-Browser verwendet werden kann: $JIFF$. Bei weiterer Betrachtung gibt es noch einen zweiten MPC-Motor für den Web-Browser, welcher auf einem populären nativen MPC-Motor aufbaut, die Web-Version von MPyC: $MPyC - WEb$.

MPC-Motoren im Web-Browser. Grundsätzlich gibt es zwei Arten komplexere Anwendungen im Web-Browser Auszuführen: (1) direkt mit JavaScript, oder (2) über WebAssembly (Wasm)[2]. JavaScript hat den Vorteil, dass es relativ einfach ist, eine Anwendung direkt in dieser Programmier-Sprache zu erstellen, welche dann z.B., via Web-Browser oder node.js[3] (JavaScript für, z.B., Server-Anwendungen) läuft. Wasm hingegen ist eine Stack-basierte virtuelle Maschine, und relativ gesehen ist es um einiges komplizierter bzw. aufwendiger Anwendung direkt in Wasm zu programmieren. Deshalb gibt es dedizierte Wasm-Compiler, welche den Quellcode von einer Programmier-Sprache nach Wasm übersetzen. Beispiele für Wasm-Compiler sind Emscripten[4] (C++ → Wasm), oder PyScript[5] / py2wasm[6] (Python → Wasm).

Bereits im Jahr 2018 haben Barak et al. mit MPC im Browser experimentiert. Im Zuge dessen wurde eine administrative Web-App erstellt, sowie auch ein dediziertes MPC-Protokoll für Teilnehmer mit geringer Bandbreite designt; genannt HyperMPC. Die Web-App bot eine benutzerfreundliche Möglichkeit, wie Teilnehmer ihre Daten eingeben und sich an der Berechnung beteiligen können (via Web-Browser / Mobilgeräte /Cloud-Instanz / IoT). In einem groß-angelegten Test, wurde mit dem Protokoll HyperMPC sogar ein Szenario mit 500 Teilnehmern realisiert. Jedoch wurde im dazugehörigen GitHub-Repository - „MATRIX: MPC Simulation Framework"[7]- nur die Web-Anwendung und Deployment-Möglichkeiten veröffentlicht; und der Kern der Anwendung – der Web-MPC-Motor – scheint nicht auffindbar zu sein.

3.1. JIFF

In diesem Teil-Abschnitt wird der Web-MPC-Motor **JIFF** (Boston Multiparty, 2024) – „JavaScript library for web-based applications that employ MPC" - näher betrachtet. Zuerst wird die grundlegende Basis beschrieben. Im Anschluss daran wird die Praxistauglichkeit von **JIFF** basierend auf folgenden Kriterien untersucht: *Relevanz*, *Bedienbarkeit*, und *Laufzeit*, im Folgenden auch als Praxis-Check bezeichnet.

[1] webassembly.org/docs/non-web
[2] webassembly.org
[3] nodejs.org
[4] emscripten.org
[5] pyscript.net
[6] wasmer.io/posts/py2wasm-a-python-to-wasm-compiler
[7] github.com/cryptobiu/MATRIX

Praxis-Check: Basis (JavaScript) & Relevanz. **JIFF** wird von "multiparty" (Boston University, USA) entwickelt, und basiert auf JavaScript. Durch die Verwendung von JavaScript kann **JIFF** direkt im Web-Browser ausgeführt werden.

Der Quellcode des MPC-Motors ist öffentlich auf GitHub[8] verfügbar; wobei die Haupt-Einstiegsseite – neben GitHub – auf „multiparty"'s Web-Seite[9] zu finden ist. Nachdem **JIFF** ein GitHub-Projekt ist, können folgende Werte für die Relevanz abgelesen werden (Stand Anfang 2024):

- ⭐ GitHub-Sterne: 244
- 📲 „GitHub-Forks" bzw. Klone des GitHub-Projekts: 51
- 💽 „Releases" bzw. Veröffentlichungen von stabilen Versionen des MPC-Motors: - (keine)

Praxis-Check: Bedienbarkeit & Dokumentation. **JIFF** stellt ein Beispiel-Projekt zur Verfügung („JIFF Standalone Application Example"[10]), welches den Einstieg vereinfacht. Für die erfolgreiche Ausführung muss einerseits ein **JIFF-Server** gestartet werden, welcher für die Kommunikation zwischen den Teilnehmern zuständig ist; und andererseits werden **JIFF-Teilnehmer** entweder direkt im Web-Browser eingebunden, oder der Code wird via, z.B., node.js ausgeführt. Beide **JIFF-Komponenten** sind in JavaScript verfasst und der Quellcode, inklusive Ausführung, ist nachvollziehbar aufgebaut.

Als Zusatz stellt „multiparty" auch eine dedizierte **JIFF-Dokumentation** zur Verfügung[11]. In dieser Dokumentation werden zahlreiche Methoden, Klassen, und „Member"-Funktionen beschrieben.

Praxis-Check: Laufzeit. Die Benchmarks vom wissenschaftlichen Artikel des populärsten generischen nativen MPC-Motors für eine beliebige Anzahl an Teilnehmern – MP-SPDZ (Keller, MP-SPDZ: A Versatile Framework for Multi-Party Computation, 2020) – vergleichen verschiedenste native MPC-Motoren auf Basis der MPC-Berechnung vom „Inner Product"[12] (InProd / Inneres Produkt bzw. Skalarprodukt). Der Vergleich mit dem InProd ist insofern für MPC-Benchmarks interessant, da es zahlreiche Multiplikationen beinhaltet; und die Addition bei MPC grundsätzlich nahezu ohne bemerkenswerten (Kommunikations-)Aufwand ausgeführt werden kann.

Bei den Benchmarks vom MP-SPDZ-Artikel wurden zwei Vektoren mit jeweils 100.000-Elementen für das InProd verwendet. Die Anzahl der Teilnehmer ist im Hauptartikel nicht beschrieben; jedoch wird im dazugehörigen GitHub-Projekt[13], welches den Quellcode für die unterschiedlichen Benchmarks beinhaltet, z.B., MPyC mit folgender Programmzeile mit einer Anzahl von 3 Teilnehmern gestartet: *`python innerprod.py -M 3`*.

Um den gleichen Benchmark auch für **JIFF** auszuführen, wurde das oben-genannte **JIFF-Beispielprojekt** erweitert. Und da der Benchmark auf einer rein kommandozeilen-basierten Linux-Server-Umgebung ausgeführt wurde, wurden die Teilnehmer via node.js gestartet. Die Ausführung via node.js sollte jedoch keine signifikanten schlechteren Laufzeiten ergeben - im Vergleich zum Web-Browser - sondern tendenziell eher bessere. Abbildung 1 zeigt den vereinfachten erstellten **JIFF-Quellcode** für die Berechnung des InProd; dabei stellen Teilnehmer 1 & 2 jeweils einen Vektor zur Verfügung und der „Threshold" (Schwellwert für das Rekonstruieren bei „Secret Sharing" (dem geheimen Teilen)) ist auf *`n-1`* eingestellt (d.h. alle Teilnehmer werden für die Rekonstruktion eines aufgeteilten Elementes benötigt. Wobei aus Implementationssicht, der „Threshold" bei **JIFF's „Share Array"** in diesem Falle auf *`n`* eingestellt wird; welches auch in der dazugehörigen Dokumentation beschrieben ist[14].

Die durchschnittliche Laufzeit von 10 Ausführungen des Benchmarks – **InProd** für **2 Vektoren** mit jeweils **100.000**-Elementen und **3 Teilnehmern** – beträgt ca. **408s** bzw. ca. **6,75min.** Wobei die Bit-Größe des Modulus bei diesem **JIFF-Programm** lediglich 24-bit beträgt (basierend auf **JIFFs** „Standalone"-Beispiel-Programm).

[8] github.com/multiparty/jiff
[9] multiparty.org/jiff
[10] github.com/multiparty/jiff-standalone-example
[11] multiparty.org/jiff/docs/jsdoc
[12] en.wikipedia.org/wiki/Dot_product
[13] github.com/mkskeller/mpc-benchmarks/blob/master/mpyc/run.sh
[14] multiparty.org/jiff/docs/jsdoc/module-jiff-client-JIFFClient.html - share_array

```
// The MPC computation for the inner product of 2 arrays
async function computation(args, jiffClient) {
  ...
  // Each player gets a share of the respective "active player array"
  ...

  // Only player 1 & 2 are sharing an input
  var senders_list = [1,2];

  // share_array
  let shares = jiffClient.share_array(input, input.length, threshold,
recv_list, senders_list);
  // Compute & Open
  try {
    // Multiply all shared input arrays element-wise
    // As only Player 1&2 provide an "active array":
    // ...only InnerProduct of `shares[1] * shares[2]`
    var array = shares[1];
    for (var i=0; i < array.length; i++) {
      array[i] = array[i].smult(shares[2][i]);
    }

    // Sum up elements
    let sum = array[0];
    for (var i=1; i < array.length; i++) {
      sum = sum.sadd(array[i]);
    }

    // Open the result -> Get the final output
    const output = await jiffClient.open(sum);
   ...
```

Abbildung 1 - Vereinfacht-dargestellter Quellcode für die Berechnung vom Skalarprodukt ("Inner Product" / InProd) mit dem MPC-Motor JIFF. Basiert auf Code-Elementen von „MPC-SoK"'s GitHub-Projekt: github.com/MPC-SoK/frameworks/blob/master/jiff/source/innerprod/mpc.js.

3.2. MPyC-Web

In diesem Teil-Abschnitt wird der Web-MPC-Motor **MPyC-Web** (Nikolov & Schoenmakers, 2024) – „MPC for Python - Web" - näher betrachtet. Zuerst wird die grundlegende Basis beschrieben. Im Anschluss daran wird die Praxistauglichkeit von **MPyC-Web** basierend auf folgenden Kriterien untersucht: Relevanz, Bedienbarkeit, und Laufzeit; im Folgenden auch als Praxis-Check bezeichnet.

Praxis-Check: Basis (Python) & Relevanz. MPyC[15] - „Multiparty Computation in Python" – wurde bzw. wird hauptsächlich von Berry Shoenmakers (Eindhoven University of Technology, Niederlande) entwickelt, und basiert auf Python. Emil Nikolov hat **MPyC** für den Gebrauch im Web-Browser erweitert: **MPyC-Web**. Um **MPyC** im Web-Browser laufen zu lassen, wird der Python-Code mittels – dem oben-genannten – PyScript nach Wasm übersetzt. Weiters wurde auch die Kommunikation zwischen den Teilnehmern adaptiert, da Sockets im Web-Browser, für

[15] win.tue.nl/~berry/mpyc

eine Peer-to-Peer-Kommunikation, nicht geeignet bzw. nicht einfach zu realisieren sind. Deshalb wurde für die Kommunikation zwischen den Teilnehmern PeerJS[16] verwendet, welches eine JavaScript-Bibliothek bzw. -API für WebRTC[17] realisiert.

Der Quellcode beider MPC-Motoren ist öffentlich auf GitHub (**MPyC**[18]; **MPyC-Web**[19]) verfügbar; wobei die Haupt-Einstiegsseite für den nativen MPC-Motor **MPyC** – neben GitHub – auf Berry Schoenmakers Web-Seite der Eindhoven University of Technology[20] zu finden ist. Nachdem **MPyC/-Web** GitHub-Projekte sind, können folgende Werte für die Relevanz abgelesen werden (Stand Anfang 2024); da **MPyC-Web** zur Zeit des Verfassens dieses Berichts relativ neu ist (Oktober 2023) werden diese Werte im jeweiligen Unterpunkt angegeben und primär nur die Werte der nativen Version gelistet:

- ⭐ GitHub-Sterne: 344
    - *(Web: 1)*
- 📲 „GitHub-Forks" bzw. Klone des GitHub-Projekts: 75
    - *(Web: - (keine))*
- 💿 „Releases" bzw. Veröffentlichungen von stabilen Versionen des MPC-Motors: 8 „Tags" bzw. Markierungen von Juni 2018 bis April 2024
    - *(Web: 7 „Tags" von Oktober 2023 bis Dezember 2023)*

Praxis-Check: Bedienbarkeit & Dokumentation. MPyC-Web stellt ebenfalls „Standalone" Beispiel-Projekte im dazugehörigen GitHub-Projekt zur Verfügung, welche den Einstieg vereinfachen. Für die erfolgreiche Ausführung muss jeder **MPyC-Web-Teilnehmer** eine Instanz des Web-MPC-Motors in einem Web-Browser-Fenster bzw. -Tab starten, und initiale Verbindungsinformationen (IDs der Teilnehmer) mit den anderen Teilnehmern austauschen; welche, z.B., auf der zur-Verfügung-gestellten Demo-Web-Seite ersichtlich sind[21]. Auch wenn es einen **MPyC-Web-Server** für die initiale Bereitstellung der Web-Seite und Web-MPC-Motors gibt, erfolgt danach der eigentliche Datenaustausch – für die MPC-Berechnung – ausschließlich über Peer-to-Peer-Verbindungen zwischen den **MPyC-Web-Teilnehmern**. Der Quellcode für die Ausführung einer MPC-Berechnung wird in Python verfasst, und ist nachvollziehbar aufgebaut.

Als Zusatz stellen Berry Schoenmakers (et al.) auch eine dedizierte **MPyC-Dokumentation** auf „Read The Docs"[22] und „GitHub-Pages"[23] zur Verfügung. In dieser Dokumentation werden zahlreiche Methoden, Klassen, und „Member"-Funktionen beschrieben. Für **MPyC-Web** hingegen gibt es keine dedizierte Dokumentation; neben dem Code und Beispiel-Programmen auf GitHub.

Praxis-Check: Laufzeit. Wie bereits beim „Praxis-Check: Laufzeit" von **JIFF** erwähnt, hat Marcel Keller im wissenschaftlichen Artikel zu MP-SPDZ zahlreiche MPC-Motoren gebenchmarkt (Keller, MP-SPDZ: A Versatile Framework for Multi-Party Computation, 2020). Darunter wurde auch die Laufzeit des „Inner Product" (InProd / Inneres Produkt bzw. Skalarprodukt) des nativen MPC-Motors **MPyC** untersucht (wie oben bei **JIFF**: 2 Vektoren mit jeweils 100.000-Elementen und 3 Teilnehmern).

Grundsätzlich ist die Laufzeit im Web-Browser langsamer, wie wenn man eine Anwendung nativ ausführt. Denn auch wenn man z.B. einen dedizierten Übersetzer von Python nach Wasm verwendet – wie den oben-genannten

[16] github.com/peers/peerjs
[17] webrtc.org
[18] github.com/lschoe/mpyc
[19] github.com/e-nikolov/mpyc-web
[20] win.tue.nl/~berry/mpyc
[21] mpyc-web.netlify.app
[22] mpyc.readthedocs.io
[23] lschoe.github.io/mpyc

py2wasm – ist die native Python-Version in der Regel performanter[24]. Abbildung 2 stellt diesen Laufzeit-Vergleich zwischen nativem Python und Wasm von Python graphisch dar. Deshalb stellt der Benchmark des MP-SPDZ-Artikels sozusagen – derzeit – eine untere Schranke für die Laufzeit von **MPyC-Web** dar. D.h. die untere Schranke für das **InProd** für **2 Vektoren** mit jeweils **100.000**-Elementen und **3 Teilnehmern** – beträgt ca. **13s** (Sekunden). Wobei die Bit-Größe des Modulus bei diesem **MPyC-Benchmark** 64-bit beträgt.

Des Weiteren kann – basierend auf den Laufzeiten von Abbildung 2 – für die obere Schranke abgeschätzt werden, dass **MPyC-Web** ca. 2- bis 4-mal langsamer ist, wie **„MPyC-Nativ"**. D.h. für die obere Schranke kann eine Laufzeit von ca. **30-60s** (Sekunden) abgeschätzt werden.

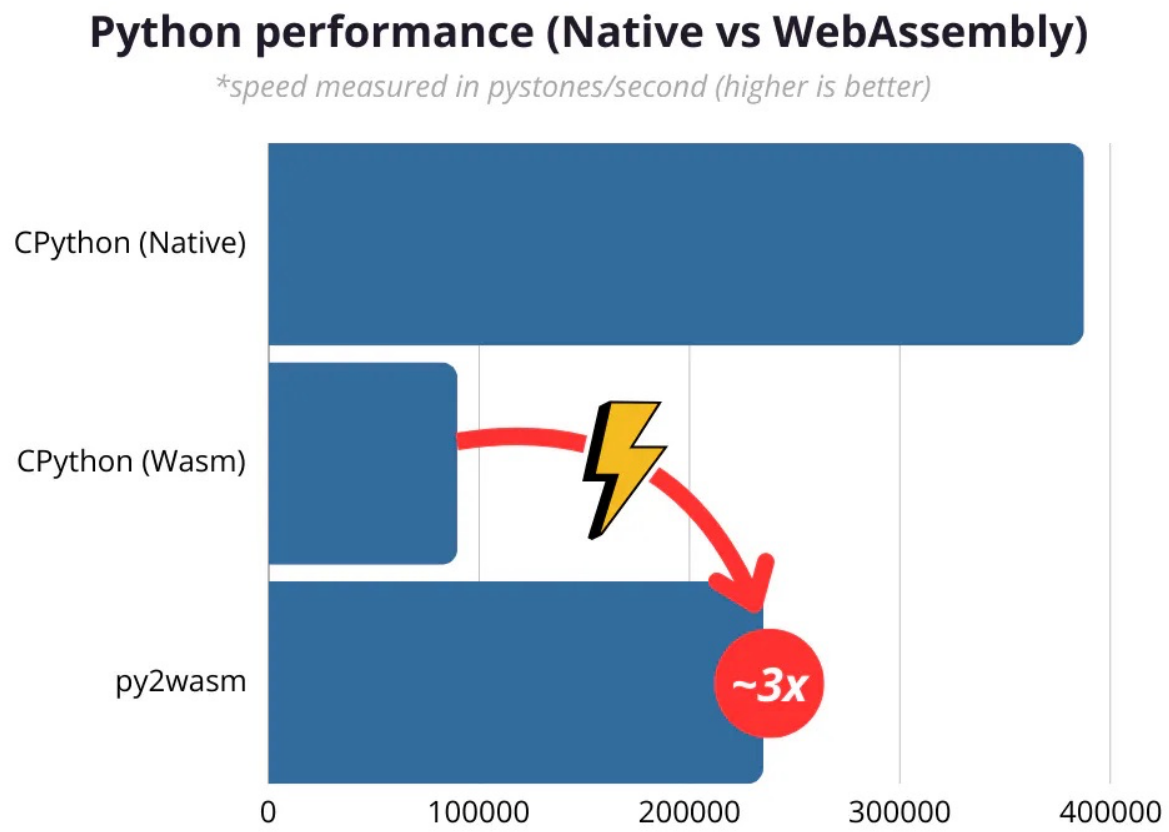

Abbildung 2 - Laufzeit-Vergleich zwischen (1) Python nativ, (2), Wasm via CPython, und (3) Wasm via py2wasm. Einerseits optimiert py2wasm die Ausführung via Wasm, und ist um bis zu ca. 3-mal performanter wie der standard Python-zu-Wasm-Übersetzer (CPython). Und andererseits ist ersichtlich, dass – zur Zeit – natives Python grundsätzlich performanter wie Wasm ist. Bild-Quelle: https://wasmer.io/posts/py2wasm-a-python-to-wasm-compiler

4. Conclusio & Weiterführende Arbeiten

In diesem Abschnitt wird zuerst der Bericht konkludiert, und im Anschluss daran Potentiale für fortführende Richtungen bzw. weiterführende Arbeiten aufgezeigt.

Einerseits wird der Einsatz von „Secure Multi-Party Computation" (MPC) stetig relevanter und dessen Berechnungs-Leistung (z.B. Laufzeit) stetig besser (siehe z.B. (Koch, 2024)). Und andererseits wird der Web-Browser, basierend auf der zunehmend komplexeren und dynamischeren Ausführungsumgebung bei End-

[24] wasmer.io/posts/py2wasm-a-python-to-wasm-compiler

Nutzern, eine stetig attraktivere Umgebung für Anwendungen, welche vor allem End-Nutzer betreffen. Deshalb gibt es bereits einige potentiell interessante und relevante Anwendungsfälle für MPC im Web-Browser (siehe z.B. Abschnitt (Potentielle) Anwendungsfälle von MPC im Web-Browser). Im Hinblick auf die Ausführungsmöglichkeit von MPC im Web-Browser, gibt es grundsätzlich zwei vielversprechende Ansätze: entweder rein via JavaScript, oder via der Übersetzung von der Ursprungs-(Programmier-)Sprache nach WebAssembly (Wasm) (siehe auch Abschnitt MPC-Web-Frameworks bzw. MPC-Web-Motoren). Jedoch gibt es aktuell weitaus mehr bekannte/populäre bzw. relevante native MPC-Motoren wie Web-MPC-Motoren. Die aktuell bekanntesten Web-MPC-Motoren sind JIFF (JavaScript) und MPyC-Web (Python→Wasm).

JIFF vs. MPyC-Web. Da **JIFF** auf reinem JavaScript aufbaut, kann es tendenziell mit weniger Aufwand in den Web-Browser eingepflegt werden. Wobei das Übersetzen von **MPyC** (Python) nach **MPyC-Web** (Wasm), und anschließende Einpflegen in den Web-Browser für, z.B., (Web-)Entwickler / IT-Forscher einen nicht allzu hohen Mehraufwand darstellen sollte.

Der Test-Benchmark für die MPC-Berechnung eines „Inner Product" (InProd / Inneres Produkt bzw. Skalarprodukt) von 2 Vektoren mit jeweils 100.000 Elementen, und 3 Teilnehmern, beträgt für

- **JIFF** ca. 6,5min (24-bit Modulus), und für
- **MPyC-Web** *(abgeschätzt)* ca. ≤30-60s bzw. ≤1min (64-bit Modulus).

D.h. nach einem ersten Benchmark bzw. nachfolgender Abschätzung ist **MPyC ca. 6-mal performanter als JIFF**. Infolgedessen ist – nach aktuellem Stand – **MPyC-Web** die bessere Wahl, um die Laufzeiten für MPC-Berechnungen *(tendenziell)* niedriger zu halten. Haben die Laufzeiten keine signifikante Relevanz beim jeweiligen Anwendungsfall, kann **JIFF** durch das einfachere Einpflegen in den Web-Browser die bessere Wahl sein.

Weitere Benchmarks. Der erste Benchmark-Vergleich für das InProd ist insofern für MPC relevant, da es zahlreiche Multiplikationen beinhaltet. Jedoch wäre es nun interessant, weitere Benchmarks mit unterschiedlichen Einstellungen durchzuführen; z.B. Größe der Vektoren beim InProd / Anzahl der Teilnehmer / Netzwerkverbindung. Vor allem in Bezug auf die Netzwerkverbindung wäre eine eingeschränkte Verbindung interessant, da es grundsätzlich realitätsnäher ist; z.B. wie die 100 Mbit/s Bandbreite und 50ms Latenz des WAN-Benchmarks[25] beim MP-SPDZ-Artikel (siehe Anhang) (Keller, MP-SPDZ: A Versatile Framework for Multi-Party Computation, 2020). Z.B. zeigten Benchmarks mit unterschiedlichen Latenzen zwischen den nativen MPC-Motoren MP-SPDZ und MPyC interessante Ergebnisse; dass etwa MPyC's *(teilweiser)* asynchroner Ansatz im Python-Code dazu führte, dass MPyC bei steigender Latenz - relativ zu MP-SPDZ gesehen –performanter wurde (Lorünser & Wohner, 2020). Des Weiteren kann auch die Leistungs-Metrik Speicherverbrauch zu interessanten Erkenntnissen führen; sowie auch der Vergleich zwischen unterschiedlichen Sicherheitsmodellen und/oder Ansätzen des geheimen Teilens.

[25] github.com/mkskeller/mpc-benchmarks/blob/master/mpyc/run.sh

Literaturverzeichnis

Barak, A., Hirt, M., Koskas, L., & Lindell, Y. (2018). An End-to-End System for Large Scale P2P MPC-as-a-Service and Low-Bandwidth MPC for Weak Participants. *CCS.* New York: https://dl.acm.org/doi/10.1145/3243734.3243801.

Goyal, V., Liu-Zhang, C.-D., & Ostrovsky, R. (2023). Asymmetric Multi-Party Computation. *Information-Theoretic Cryptography.* Aarhus (Denmark): https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.ITC.2023.6.

Nikolov, E., & Schoenmakers, B. (2024). *MPyC-Web*. Retrieved from GitHub: https://github.com/e-nikolov/mpyc-web

Schoenmakers, B. (2024). *MPyC: Multiparty Computation in Python*. Retrieved from GitHub: github.com/lschoe/mpyc

Boston Multiparty. (2024). *JIFF: JavaScript library for web-based applications that employ secure multi-party computation (MPC)*. Retrieved from GitHub: https://github.com/multiparty/jiff

Koch, K. W. (2024). *A-SIT Technologie.* Retrieved from Evaluierung von MPC's Stand der Technik: https://technology.a-sit.at/evaluierung-von-mpcs-stand-der-technik/

Keller, M. (2020). MP-SPDZ: A Versatile Framework for Multi-Party Computation. *CCS.* New York: https://dl.acm.org/doi/10.1145/3372297.3417872.

Lorünser, T., & Wohner, F. (2020). Performance Comparison of Two Generic MPC-frameworks with Symmetric Ciphers. *SECRYPT.* Paris (France): https://www.scitepress.org/Link.aspx?doi=10.5220/0009831705870594.