A-SIT Technology
E-Government Innovation Center

14 June 2024 Analysis · Privacy & Crypto · Web & Cloud · 21 min read

Evaluation of Secure-Multi-Party-Computation-Frameworks for the Web Browser

Karl W. Koch



Authors	Karl W. Koch
Version	2024-03-31
Source	https://technology2.a-sit.at/en/publications/evaluation-of-secure-multi-party-computation-frameworks-for-the-web-browser/
Note	Machine translation.

Key findings

- JIFF (pure JavaScript) and MPyC-Web (Python translated to WebAssembly) are currently the best-known MPC web engines.
- In the inner product benchmark (2 vectors of 100,000 elements, 3 participants), JIFF took about 408 s on average with a 24-bit modulus.
- For MPyC-Web, an estimate gives about 30–60 s with a 64-bit modulus; MPyC is thus about 6 times faster than JIFF.
- Being pure JavaScript, JIFF is easier to integrate into the web browser and can be the better choice where runtime is not critical.

- MPC in the web browser lowers the entry barrier because participants need not install anything; use cases include hybrid MPC engines and IoT.

Secure Multi-Party Computation (MPC) is now a practical privacy-preserving cryptographic building block. With the help of MPC, numerous data analyses can be carried out in such a way that the participants' input data remains secret and privacy is maintained. For instance, a survey can be conducted in a meeting/lecture where only the result is known and cryptographically – at protocol level – it is ensured that practically no identifiable conclusions can be drawn about the respective participants' input data. And ideally, the participants actively take part in the computation.

One of the next steps in enabling MPC for a large number of end users – as in the example above – is to run the MPC frameworks on the corresponding end devices. Especially because the computer usage environment has become very dynamic, and the trend continues in this direction: Computer/SmartPhone/Tablet/SmartWatch/SmartHome/etc. Hence, the execution in the web browser becomes more and more relevant for respective applications.

Therefore, in this project (1) current approaches for running MPC in the web browser are examined, and (2) two specific MPC frameworks (JIFF and MPyC-Web) are then examined in more detail and compared based on their practical suitability for running MPC in the web browser.

1. Introduction

"Secure Multi-Party Computation" (MPC) is now a practical privacy-preserving cryptographic building block. With the help of MPC, numerous data analyses and computations in general can be carried out in such a way that the participants' input data remains secret and, so to speak, privacy is maintained. For instance, a survey/vote ("voting") can be conducted spontaneously and in real time in a meeting/session/lecture where only the result is known and, cryptographically/mathematically – at protocol level – it is ensured that practically no identifiable conclusions can be drawn about the respective participants. Taking part in such a survey/vote by means of MPC has the advantage that, on the one hand, nobody else learns the respective person's input – which can tend to lead to more open and honest answers – and, on the other hand, it can be ensured cryptographically/mathematically that the participating persons have entered or voted with correct data. And ideally, in order to maximise security during the computation, the respective parties that provide data actively take part in the computation (= are participants).

MPC entry barriers. To actively take part in a joint MPC computation, or even just to experiment, the respective MPC framework (hereinafter MPC engine) must be "set up". For example, all required software packages or libraries must be installed, the network must be configured and, finally, the MPC program must be brought to successful execution for every participant – in the respective system environment. Although many MPC engines provide Docker containers with easily accessible tutorial programs, it is still an initial hurdle to successfully "set up" or activate everything for the respective experiments and computations. This entry barrier is particularly high for those who are not familiar with MPC engines and/or when the participants are not technical experts or not tech-savvy.

One of the next steps in enabling MPC for a large number of end users or potential participants – as in the example above – is the simple execution of the MPC engines. The aim is to make the potential entry barrier as small as possible and, ideally, to require nothing to be installed or "set up" by the participants themselves. Moreover, this entry barrier must be kept minimal for all potential end devices of the participants. This applies especially because the computer usage environment has become very dynamic, and the trend continues in this direction. People increasingly own a multitude of technical devices that could be considered as end devices for MPC: computer / laptop / smartphone / tablet / smartwatch / smart home devices / etc. For example, some people may take part in the above-mentioned survey/vote with a computer/laptop, while others can only take part with a smartphone or tablet.

Why MPC in the web browser? One way of enabling MPC on a large number of end devices without participants having to set up the MPC engines themselves is platform-specific applications (apps). That is, there would be apps for Android / iOS / iPad / Linux / Windows / macOS / etc. for the participants. In this case, however, an app has to be created for each platform in the respective programming language. And although there are by now some approaches for creating an app in only one programming language or framework so that the app can then be used on several end devices, creating and maintaining such an app for all potential end devices is still a considerable additional effort.

What connects nearly all end devices in this context – or, in short, what appeals to most people in the scenarios mentioned – is access to a web browser. Provided that the web browser version of, e.g., Safari, Firefox, Edge or Chrome is up to date on the respective person's device, a fundamentally homogeneous execution environment is given across, e.g., laptop, smartphone and tablet. Thanks to this homogeneous execution environment spanning numerous end devices, the use of the web browser is becoming steadily more interesting and relevant for numerous applications.

Particularly in the field of MPC, the web browser has the advantage over system or mobile applications that participants do not have to install anything; they "merely" need a compatible web browser. And since no installation is required, the "set-up" is considerably smoother for the participants. This smooth set-up makes it easier to make the added value of MPC – e.g., with regard to privacy – accessible to every person. Thus, MPC in the web browser enables even non-experts or participants who are not tech-savvy to enter their data or cast their vote in real time via MPC; as, e.g., in the scenarios mentioned above.

Furthermore, as early as 2018, Barak et al. (Barak, Hirt, Koskas, & Lindell, 2018) reported the same relevance of making MPC available to a large number of people. In other words, MPC should become a "standard product", like secure network traffic on the internet via TLS ("Transport Layer Security"; e.g., indicated in the web browsers Firefox and Safari by the lock symbol in the address bar). Therefore, in their experiment, Barak et al. provided several ways to actively take part in an MPC computation, including via a web browser. Barak et al. also explain the importance and advantage of the web browser for participants:

"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." (Barak, Hirt, Koskas, & Lindell, 2018) and "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." (Barak, Hirt, Koskas, & Lindell, 2018)

Goals & outlook. Therefore, this project first lists (potential) use cases of MPC in the web browser and then describes how these MPC web frameworks can be executed in the web browser. In the course of this, two specific MPC frameworks or engines are examined more closely and compared based on their practical suitability: JIFF and MPyC-Web. Finally, a conclusion & further work section briefly summarises the findings and points out potentially interesting directions for further work.

2. (Potential) Use cases of MPC in the web browser

This section lists further potentially interesting and relevant use cases for running MPC in the web browser. On the one hand, the previous section already mentioned an example with active participants in a meeting/lecture/session. Derived from this, MPC in the web browser – owing to its simplicity – can in general favour or enable numerous use cases. Furthermore, MPC in the web browser is of particular interest, e.g., in hybrid applications (dedicated server and a relatively easy-to-use web interface) and "IoT" scenarios.

Universal: easy MPC access. In theory, wherever a person or organisation wants to take an active part in an MPC computation, execution via a web browser can be helpful. As described in the A-SIT report "Evaluation of the State of the Art of MPC" from early 2024 (Koch, 2024), MPC is becoming increasingly practical and, as a consequence, is also used more frequently. If the MPC entry barrier is now reduced to a minimum, and execution is thus made possible for a large number of potential participants, then on the one hand more persons or organisations can actively take part, and on the other hand additional use cases may be enabled by this simplicity.

Hybrid MPC engines. Just as there are hybrid drives with electricity and diesel/petrol in cars, a "hybrid MPC engine" use case can be imagined as follows: one participant takes part via a native MPC engine or application (e.g., a dedicated server in a Linux environment via MPyC (Schoenmakers, MPyC: Multiparty Computation in Python, 2024)), and another participant takes part via a web MPC engine in the web browser (e.g., MPyC-Web (Nikolov & Schoenmakers, 2024)). A practically relevant scenario here can be inference in machine learning (ML). In this case, the participant with the native MPC engine provides an ML model, and the participant with the web MPC engine enters data for the ML evaluation and only has access to a web browser for the MPC computation. The added value of this asymmetric or hybrid use case is, on the one hand, the accessibility through the web browser and, on the other hand, the generally higher computing capacity of the native MPC engine. For example, Goyal, Liu-Zhang and Ostrovsky investigated this sub-area of MPC in 2023 and called it "Asymmetric MPC" (Goyal, Liu-Zhang, & Ostrovsky, 2023). And, as usual with ML evaluations via MPC, the input data of the evaluating participant remains secret, and the ML model parameters of the server also remain secret.

Furthermore, use cases with several participants and hybrid MPC engines can also be relevant for offering the participants the option to choose between MPC engines in a native environment and in the web browser. Thus, e.g., some participants of the example above – in the meeting/presentation/session – could enter their data via a native MPC engine in Linux, and the other participants via a web MPC engine on the smartphone.

IoT et al. Beyond that, it can also be interesting and relevant to support further platforms beyond the web browser. If the MPC engine is executed in the web browser via WebAssembly (Wasm), it is in principle also possible to run the MPC engine on a wide variety of "Internet of Things" (IoT) devices[1]. It would then be

possible, for example, for several IoT devices to jointly compute a function via MPC, with each device learning only the result. For example, if the traffic light system at several intersections were to determine the traffic load in an MPC computation.

3. MPC web frameworks or MPC web engines

This section first explains in general the approaches to MPC frameworks or engines on the web, and then takes a closer look at two MPC web engines – based on their practical suitability: JIFF and the web version of MPyC (MPyC-Web).

State of the art of MPC engines. Based on the A-SIT report "Evaluation of the State of the Art of MPC" from 2024 (Koch, 2024), there are in principle two classifications of MPC engines: (1) how many parties can actively take part in the MPC computation (= number of participants), and (2) for which kind of MPC computations the respective MPC engine was specialised (= e.g., generic or for privacy-preserving ML). The A-SIT report mentioned above also listed the currently most up-to-date and relevant MPC engines. It contained only one MPC engine that can be used directly in a web browser: JIFF. On closer inspection, there is a second MPC engine for the web browser, which builds on a popular native MPC engine, the web version of MPyC: MPyC-Web.

MPC engines in the web browser. In principle, there are two ways of running more complex applications in the web browser: (1) directly with JavaScript, or (2) via WebAssembly (Wasm)[2]. JavaScript has the advantage that it is relatively easy to create an application directly in this programming language, which then runs, e.g., via the web browser or node.js[3] (JavaScript for, e.g., server applications). Wasm, in contrast, is a stack-based virtual machine, and it is comparatively much more complicated and laborious to program applications directly in Wasm. Therefore, there are dedicated Wasm compilers that translate the source code from a programming language to Wasm. Examples of Wasm compilers are Emscripten[4] (C++ → Wasm) or PyScript[5] / py2wasm[6] (Python → Wasm).

As early as 2018, Barak et al. experimented with MPC in the browser. In the course of this, an administrative web app was created and a dedicated MPC protocol for participants with low bandwidth was designed, called HyperMPC. The web app offered a user-friendly way for participants to enter their data and take part in the computation (via web browser / mobile devices / cloud instance / IoT). In a large-scale test, a scenario with 500 participants was even realised with the HyperMPC protocol. However, the associated GitHub repository – "MATRIX: MPC Simulation Framework"[7] – published only the web application and deployment options; and the core of the application – the web MPC engine – does not appear to be findable.

3.1. JIFF

This subsection takes a closer look at the web MPC engine JIFF (Boston Multiparty, 2024) – "JavaScript library for webbased applications that employ MPC". First, the basic foundation is described. Subsequently, the practical suitability of JIFF is examined based on the following criteria: relevance, usability and runtime; hereinafter also referred to as the practicality check.

Practicality check: basis (JavaScript) & relevance. JIFF is developed by "multiparty" (Boston University, USA) and is based on JavaScript. By using JavaScript, JIFF can be executed directly in the web browser.

The source code of the MPC engine is publicly available on GitHub[8]; the main entry page – besides GitHub – can be found on the "multiparty" website[9]. Since JIFF is a GitHub project, the following relevance values can be read off (as of early 2024):

- GitHub stars: 244
- "GitHub forks", i.e. clones of the GitHub project: 51
- "Releases", i.e. publications of stable versions of the MPC engine: - (none)

Practicality check: usability & documentation. JIFF provides an example project ("JIFF Standalone Application Example"[10]) that simplifies getting started. For successful execution, on the one hand a JIFF server must be started, which is responsible for the communication between the participants; and on the other hand, JIFF participants are either embedded directly in the web browser, or the code is executed via, e.g., node.js. Both JIFF components are written in JavaScript, and the source code, including its execution, is clearly structured.

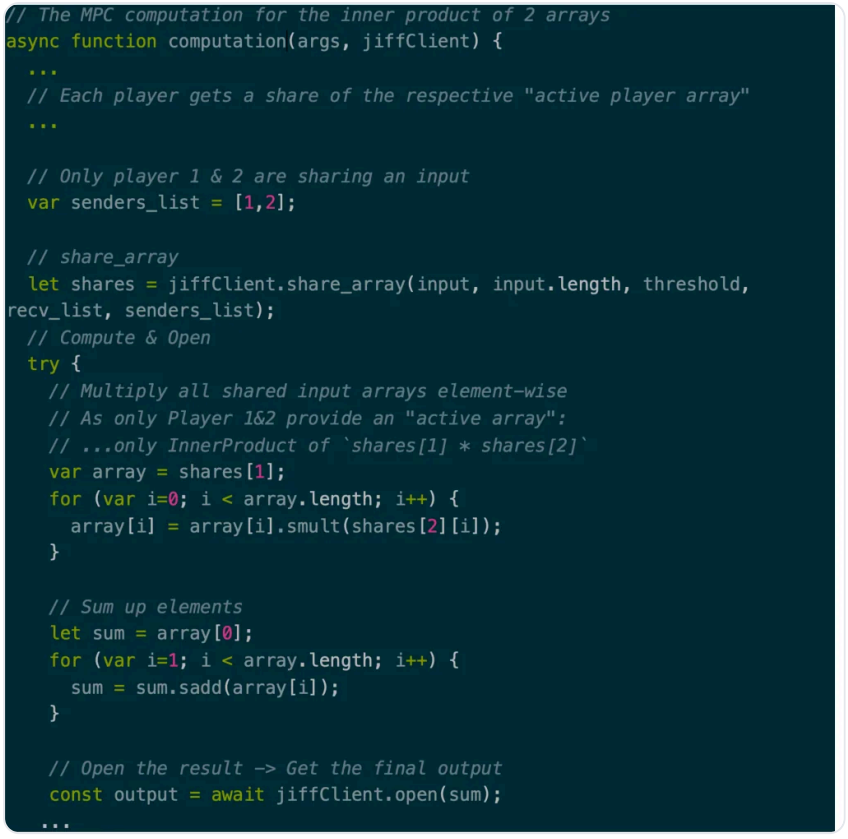
In addition, "multiparty" also provides dedicated JIFF documentation[11]. This documentation describes numerous methods, classes and "member" functions.

Practicality check: runtime. The benchmarks of the scientific article on the most popular generic native MPC engine for an arbitrary number of participants – MP-SPDZ (Keller, MP-SPDZ: A Versatile Framework for Multi-Party Computation, 2020) – compare a wide variety of native MPC engines on the basis of the MPC computation of the "inner product"[12] (InProd / dot product). Comparison with the InProd is interesting for MPC benchmarks because it involves numerous multiplications; and addition in MPC can in principle be carried out with almost no notable (communication) overhead.

In the benchmarks of the MP-SPDZ article, two vectors with 100,000 elements each were used for the InProd. The number of participants is not described in the main article; however, the associated GitHub project[13], which contains the source code for the different benchmarks, starts, e.g., MPyC with the following program line with 3 participants: `python innerprod.py -M 3`.

To run the same benchmark for JIFF as well, the above-mentioned JIFF example project was extended. And since the benchmark was run in a purely command-line-based Linux server environment, the participants were started via node.js. Execution via node.js should not, however, result in significantly worse runtimes compared with the web browser, but rather tend to be better. Figure 1 shows the simplified JIFF source code created for the computation of the InProd; participants 1 & 2 each provide one vector, and the "threshold" (threshold for reconstruction in "secret sharing") is set to `n-1` (i.e. all participants are needed to reconstruct a shared element). From an implementation point of view, the "threshold" of JIFF's "share array" is set to `n` in this case; this is also described in the associated documentation[14].

The average runtime of 10 executions of the benchmark – InProd for 2 vectors with 100,000 elements each and 3 participants – is approx. 408s, or approx. 6.75min. The bit size of the modulus in this JIFF program is only 24 bits (based on JIFF's "standalone" example program).



```
// 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);
  ...
```

Figure 1: Simplified source code for computing the inner product ('Inner Product' / InProd) with the MPC engine JIFF. Based on code elements from the GitHub project of 'MPC-SoK': github.com/MPC-SoK/frameworks/blob/master/jiff/source/innerprod/mpc.js.

3.2. MPyC-Web

This subsection takes a closer look at the web MPC engine MPyC-Web (Nikolov & Schoenmakers, 2024) – "MPC for Python - Web". First, the basic foundation is described. Subsequently, the practical suitability of MPyC-Web is examined based on the following criteria: relevance, usability and runtime; hereinafter also referred to as the practicality check.

Practicality check: basis (Python) & relevance. MPyC[15] – "Multiparty Computation in Python" – was and is mainly developed by Berry Schoenmakers (Eindhoven University of Technology, Netherlands) and is based on Python. Emil Nikolov extended MPyC for use in the web browser: MPyC-Web. To run MPyC in the web browser, the Python code is translated to Wasm using PyScript, mentioned above. Furthermore, the communication between the participants was adapted, since sockets in the web browser are not suitable, or not easy to realise, for peer-to-peer communication. Therefore, PeerJS[16], which implements a JavaScript library or API for WebRTC[17], was used for the communication between the participants.

The source code of both MPC engines is publicly available on GitHub (MPyC[18]; MPyC-Web[19]); the main entry page for the native MPC engine MPyC – besides GitHub – can be found on Berry Schoenmakers' website at Eindhoven University of Technology[20]. Since MPyC/-Web are GitHub projects, the following relevance values can be read off (as of early 2024); because MPyC-Web is relatively new at the time of writing this report (October 2023), its values are given in the respective sub-item and primarily only the values of the native version are listed:

- GitHub stars: 344
    - (Web: 1)
- "GitHub forks", i.e. clones of the GitHub project: 75
    - (Web: - (none))
- "Releases", i.e. publications of stable versions of the MPC engine: 8 "tags", i.e. markers from June 2018 to April 2024
    - (Web: 7 "tags" from October 2023 to December 2023)

Practicality check: usability & documentation. MPyC-Web likewise provides "standalone" example projects in the associated GitHub project, which simplify getting started. For successful execution, each MPyC-Web participant must start an instance of the web MPC engine in a web browser window or tab and exchange initial connection information (IDs of the participants) with the other participants; this is visible, e.g., on the demo web page provided[21]. Even though there is an MPyC-Web server for the initial delivery of the web page and the web MPC engine, the actual data exchange – for the MPC computation – afterwards takes place exclusively via peer-to-peer connections between the MPyC-Web participants. The source code for executing an MPC computation is written in Python and is clearly structured.

In addition, Berry Schoenmakers (et al.) also provide dedicated MPyC documentation on "Read The Docs"[22] and "GitHub Pages"[23]. This documentation describes numerous methods, classes and "member" functions. For MPyC-Web, in contrast, there is no dedicated documentation besides the code and example programs on GitHub.

Practicality check: runtime. As already mentioned in the "Practicality check: runtime" of JIFF, Marcel Keller benchmarked numerous MPC engines in the scientific article on MP-SPDZ (Keller, MP-SPDZ: A Versatile Framework for Multi-Party Computation, 2020). Among other things, the runtime of the "inner product" (InProd / dot product) of the native MPC engine MPyC was examined (as above for JIFF: 2 vectors with 100,000 elements each and 3 participants).

In principle, the runtime in the web browser is slower than when an application is executed natively. Even when using a dedicated translator from Python to Wasm – such as the above-mentioned py2wasm – native Python is generally more performant[24]. Figure 2 shows this runtime comparison between native Python and Wasm from Python graphically. The benchmark of the MP-SPDZ article therefore currently represents, so to speak, a lower bound for the runtime of MPyC-Web. That is, the lower bound for the InProd for 2 vectors with 100,000 elements each and 3 participants is approx. 13s (seconds). The bit size of the modulus in this MPyC benchmark is 64 bits.

Furthermore, based on the runtimes in Figure 2, the upper bound can be estimated: MPyC-Web is approx. 2 to 4 times slower than "MPyC native". That is, a runtime of approx. 30-60s (seconds) can be estimated for the upper bound.

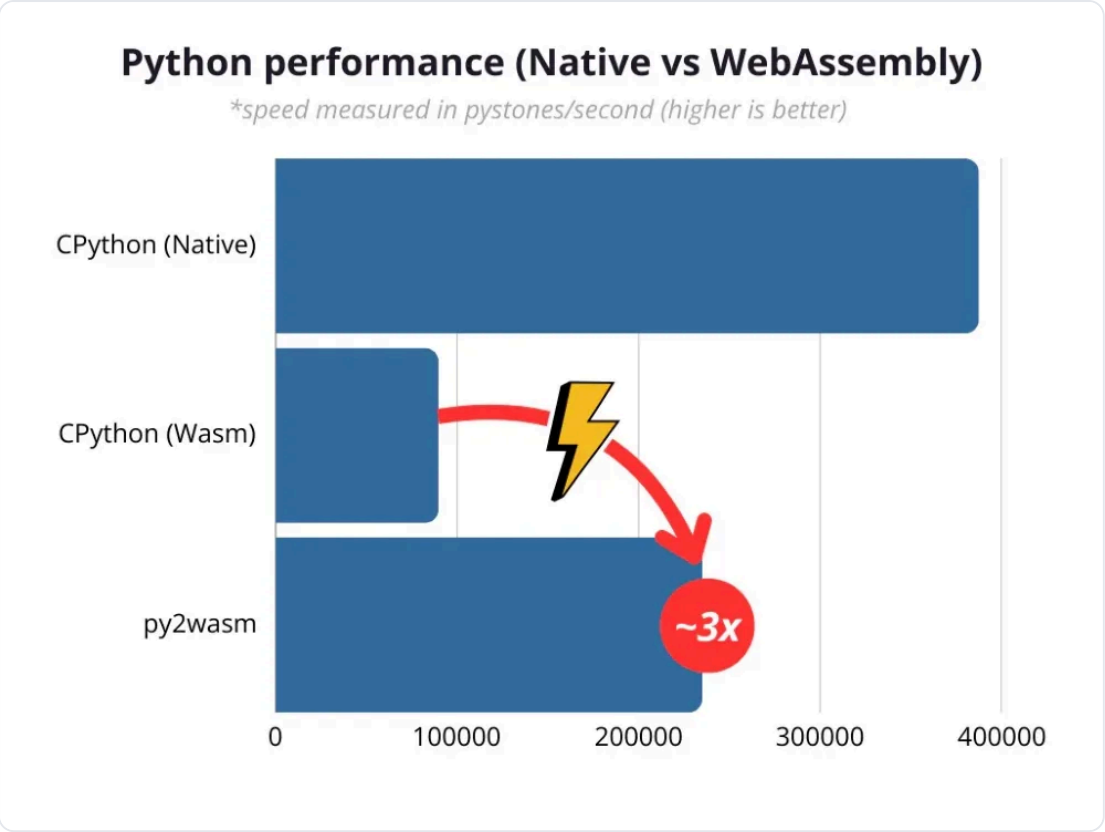


Figure 2: Runtime comparison between (1) native Python, (2) Wasm via CPython, and (3) Wasm via py2wasm. On the one hand, py2wasm optimises execution via Wasm and is up to approx. 3 times more performant than the standard Python-to-Wasm translator (CPython). On the other hand, it is evident that – at present – native Python is generally more performant than Wasm. Image source: https://wasmer.io/posts/py2wasm-a-python-to-wasm-compiler

4. Conclusion & further work

This section first concludes the report and then points out potential directions for further work.

On the one hand, the use of "Secure Multi-Party Computation" (MPC) is becoming steadily more relevant and its computational performance (e.g., runtime) is steadily improving (see, e.g., (Koch, 2024)). On the other hand, owing to the increasingly complex and dynamic execution environment of end users, the web browser is becoming an ever more attractive environment for applications that primarily concern end users. Therefore, there are already some potentially interesting and relevant use cases for MPC in the web browser (see, e.g., the section (Potential) Use cases of MPC in the web browser). With regard to the possibility of running MPC in the web browser, there are in principle two promising approaches: either purely via JavaScript, or via translation from the original (programming) language to WebAssembly (Wasm) (see also the section MPC web frameworks or MPC web engines). However, there are currently far more well-known/popular and relevant native MPC engines than web MPC engines. The currently best-known web MPC engines are JIFF (JavaScript) and MPyC-Web (Python → Wasm).

JIFF vs. MPyC-Web. Since JIFF is built on pure JavaScript, it can tend to be integrated into the web browser with less effort. Translating MPyC (Python) to MPyC-Web (Wasm) and subsequently integrating it into the web browser, on the other hand, should not represent an excessively high additional effort for, e.g., (web) developers / IT researchers.

The test benchmark for the MPC computation of an "inner product" (InProd / dot product) of 2 vectors with 100,000 elements each and 3 participants is, for

- JIFF approx. 6.5min (24-bit modulus), and for
- MPyC-Web (estimated) approx. ≤30-60s, i.e. ≤1min (64-bit modulus).

That is, according to a first benchmark and the subsequent estimate, MPyC is approx. 6 times more performant than JIFF. Consequently, as things stand, MPyC-Web is the better choice for (tending to) keep the runtimes of MPC computations lower. If runtimes are not of significant relevance for the respective use case, JIFF can be the better choice owing to its simpler integration into the web browser.

Further benchmarks. The first benchmark comparison for the InProd is relevant for MPC because it involves numerous multiplications. However, it would now be interesting to run further benchmarks with different settings; e.g., size of the vectors in the InProd / number of participants / network connection. With regard to the network connection in particular, a restricted connection would be interesting since it is generally closer to reality; e.g., like the 100 Mbit/s bandwidth and 50ms latency of the WAN benchmark[25] in the MP-SPDZ article (see appendix) (Keller, MP-SPDZ: A Versatile Framework for Multi-Party Computation, 2020). For example, benchmarks with different latencies between the native MPC engines MP-SPDZ and MPyC showed interesting results; for instance, MPyC's (partly) asynchronous approach in the Python code led to MPyC becoming more performant – relative to MP-SPDZ – as latency increased (Lorünser & Wohner, 2020). Furthermore, the performance metric of memory consumption can also lead to interesting insights, as can the comparison between different security models and/or secret sharing approaches.

References

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/veroeffentlichungen/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.

---

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 ↩
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 ↩
15. win.tue.nl/~berry/mpyc ↩
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 ↩
24. wasmer.io/posts/py2wasm-a-python-to-wasm-compiler ↩
25. github.com/mkskeller/mpc-benchmarks/blob/master/mpyc/run.sh ↩