



universität  
**uulm**



# Confidential Computing via Multiparty Computation and Trusted Computing

**Juri Dispan**

945106

**Master's Thesis**

VS-2023-05M

**Examined by**

Prof. Dr. rer. nat. Frank Kargl

Prof. Dr.-Ing. Franz J. Hauck

**Supervised by**

M.Sc. Dominik Meißner

Dr. rer. nat. Benjamin Erb

Institute of Distributed Systems  
Faculty of Engineering, Computer Science and Psychology  
Ulm University

26 April 2023



© 2023 Juri Dispan

Issued: 26th April 2023



This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike License.

To view a copy of this license, visit

<https://creativecommons.org/licenses/by-nc-sa/4.0/> or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

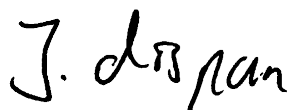
I hereby declare that this thesis titled:

**Confidential Computing via Multiparty Computation and Trusted Computing**

is the product of my own independent work and that I have used no sources or materials other than those specified. The passages taken from other works, either verbatim or paraphrased in the spirit of the original quote, are identified in each individual case by indicating the source.

I further declare that all my academic work was written in line with the principles of proper academic research according to the official "Satzung der Universität Ulm zur Sicherung guter wissenschaftlicher Praxis" (University Statute for the Safeguarding of Proper Academic Practice).

Ulm, 26 April 2023

A handwritten signature in black ink, appearing to read "J. Dispan". The signature is written in a cursive, slightly slanted style.

Juri Dispan, student number 945106



## ABSTRACT

---

In the wake of the social sciences' so-called replication crisis, researchers increasingly strive to adopt methods preventing questionable research practices in empirical studies, e. g., study preregistration and full publication of survey datasets. However, publication of survey responses poses a serious threat to the privacy of study participants. Previous work has addressed this issue while maintaining protection against questionable research practices, but either relies on *Trusted Execution Environments* (TEEs), which have been shown to be susceptible to various kinds of attacks, or on *Secure Multiparty Computation* (SMPC), requiring a honest majority of participating parties. In this work, we combine TEEs with SMPC in a platform for conducting empirical studies that provides strong guarantees for the privacy of participants. Survey responses are split into secret shares, which are distributed among a number of TEE-protected computation parties. Statistical analysis of responses is performed as an SMPC. The platform is secure against a wider range of attackers than related work, i. e., against attackers either able to circumvent the utilised TEE or controlling a majority of the computation parties. We implement a prototype of this platform and evaluate its computational performance against alternative approaches. We show that it is suitable for conducting real-world privacy-preserving empirical studies, placing only minimal computational load on survey participants. Its performance in conducting statistical analysis is inferior to its alternatives, requiring  $\approx 10$  min for performing one two-sample t-test. However, we argue that this is sufficient for real-world settings. Additionally, we list several approaches with which performance can be enhanced.



## CONTENTS

---

<b>Abstract</b>	<b>v</b>
<b>Contents</b>	<b>vii</b>
<b>List of Figures</b>	<b>ix</b>
<b>List of Tables</b>	<b>x</b>
<b>Acronyms</b>	<b>xi</b>
<b>I Background</b>	<b>1</b>
<b>1 Introduction</b>	<b>3</b>
<b>2 Fundamentals</b>	<b>7</b>
2.1 Secret Sharing . . . . .	7
2.2 Secure Multiparty Computation . . . . .	9
2.3 Trusted Execution Environments . . . . .	15
<b>3 Enabling Technologies</b>	<b>23</b>
3.1 Enabling Secure Multiparty Computation . . . . .	23
3.2 Enabling Intel SGX . . . . .	25
<b>4 Related Work</b>	<b>29</b>
4.1 Improving SMPC Performance through TEE . . . . .	29
4.2 Enhancing Privacy in Empirical Studies . . . . .	30
<b>II Contribution</b>	<b>33</b>
<b>5 Approach</b>	<b>35</b>
5.1 Using SGX for Hardening the Privacy of Empirical Studies . . . . .	35
5.2 Using SMPC for Hardening the Privacy of Empirical Studies . . . . .	36
5.3 Main Contribution: Combining TEEs and SMPC . . . . .	39
<b>6 Implementation</b>	<b>45</b>
6.1 Requirements . . . . .	45
6.2 Choice of Technology . . . . .	45
<b>7 Evaluation</b>	<b>49</b>
7.1 Methodology . . . . .	49
7.2 Results . . . . .	52
<b>8 Discussion</b>	<b>61</b>
8.1 Findings . . . . .	61
8.2 Limitations & Future Work . . . . .	66

CONTENTS

<b>9</b>	<b>Summary &amp; Conclusion</b>	<b>69</b>
9.1	Summary . . . . .	69
9.2	Conclusion . . . . .	69
	<b>Bibliography</b>	<b>71</b>



LIST OF FIGURES

---

2.1	General Architecture for Trusted Computing . . . . .	16
2.2	Life-Cycle of an SGX Enclave . . . . .	19
3.1	General Architecture of Jiff . . . . .	24
5.1	Publishing and Conducting a Study through PeQES . . . . .	37
5.2	Publishing a Study through an SMPC-based Platform . . . . .	38
5.3	Conducting a Study through an SMPC-based Platform . . . . .	39
7.1	Architectures of the Implemented Prototypes . . . . .	50
7.2	Results of Experiment E1: Compute Times . . . . .	54
7.3	Results of Experiment E1: Open Times . . . . .	55
7.4	Results of Experiment E3 . . . . .	58
8.1	Results of Profiling SMPC-SGX . . . . .	62

LIST OF TABLES

---

5.1	Attack Scenarios on Platforms for Securely Conducting Empirical Studies . . . . .	43
7.1	Results of Experiment E1: Compute Times . . . . .	55
7.2	Results of Experiment E2 . . . . .	56
7.3	Results of Experiment E3 . . . . .	57
7.4	Results of Experiment E4 . . . . .	59

## ACRONYMS

---

**AEX** *Asynchronous Exit*

**DSL** *Domain-Specific Language*

**EDMM** *Enclave Dynamic Memory Management*

**EPC** *Enclave Page Cache*

**EPCM** *Enclave Page Cache Map*

**HE** *Homomorphic Encryption*

**LE** *Launch Enclave*

**PeQES** *Platform for privacy-enhanced Qualitative Empirical Studies*

**PRM** *Processor-Reserved Memory*

**SDK** *Software Development Kit*

**SECS** *SGX Enclave Control Structure*

**SGX** *Software Guard Extensions*

**SMPC** *Secure Multiparty Computation*

**TCB** *Trusted Computing Base*

**TEE** *Trusted Execution Environment*

**TTP** *Trusted Third Party*



**Part I**

**Background**



## INTRODUCTION

---

*Statisticians should take appropriate measures to prevent their data from being published or otherwise released in a form that would allow any subject's identity to be disclosed or inferred.*

— INTERNATIONAL STATISTICAL INSTITUTE (1985),  
DECLARATION ON PROFESSIONAL ETHICS [56, §4.6]

Science largely relies on empirical methods for advancing knowledge: scientists make hypotheses and collect data by making observations. If the data suggest one or more hypotheses to be true with large enough probability, these hypotheses are accepted; the others are rejected. This process is called an *empirical study*.

In recent years, psychology as well as other social sciences have suffered a so-called *replication crisis* [106]. It was discovered that a surprisingly large number of results could not be reproduced, i. e., hypotheses accepted through previous studies did not hold true when applied to new observations [90]. Additionally, cases of scientific fraud [19, 117] and questionable research practices, e. g., *HARKing* [62], in which hypotheses are formed specifically to fit previously collected data, and *p-hacking* [107, 51], in which the significance of results is artificially raised through bogus use of statistical methods, became known [57, 116].

In order to be able to conduct more robust studies, an open research process has been suggested by the Center for Open Science. This process aims to prevent questionable research practices and ensure scientific quality. The idea of this process is that researchers ought to publicly *preregister* studies, i. e., announce their hypotheses, methodology, planned sample size and analyses prior to collecting data. After data have been collected and analyses have been performed, researchers should make these data publicly available. This process prevents a range of questionable research practices and improves reproducibility. It is aimed at restoring trust in scientific results, as it enables the scientific community to verify that no *p-hacking* or *HARKing* has occurred [106].

However, as social sciences by definition study the nature of human beings, their research process necessarily requires observation of such. In psychology, data collection is regularly performed through the usage of surveys, in which subjects self-report aspects of their inner and outer condition. Needless to say, such self-reports contain (sometimes highly) personal data. In order to raise subjects' willingness to perform self-reporting and in order to comply with legal requirements, data are mostly collected in an anonymised way, in which obvious personal identifiers such as name or street address are not captured or removed as soon as possible [42, 47].

*Anonymity alone is by no means a guarantee of confidentiality. A particular configuration of attributes can, like a fingerprint, frequently identify its owner beyond reasonable doubt.*

— INTERNATIONAL STATISTICAL INSTITUTE (1985),  
DECLARATION ON PROFESSIONAL ETHICS [56, §4.6]

It has, however, been shown that this is insufficient for making linkage of survey responses to specific persons impossible. In many cases, a combination of a small number data points from a survey response suffices to identify the responding individual, even if these data points contain no personally identifiable information by themselves [42]. This is true for demographic attributes<sup>1</sup>, but also non-demographic and seemingly benign ones such as writing style [87] or movie preferences [86]. Published datasets may thus be subject to de-anonymisation attacks even if all but the target data are removed. In spite of this fact, willingness among researchers to share datasets grows [42].

*The widespread use of computers is often regarded as a threat to individuals and organisations because it provides new methods of disclosing and linking identified records. On the other hand, the statistician should attempt to exploit the impressive capacity of computers to disguise identities and to enhance data security.*

— INTERNATIONAL STATISTICAL INSTITUTE (1985),  
DECLARATION ON PROFESSIONAL ETHICS [56, §4.6]

In light of the potentially grievous privacy impact of data publication, different approaches for ensuring scientific integrity have been proposed. Meißner et al. [81] propose a workflow that depends on a *Platform for privacy-enhanced Qualitative Empirical Studies* (PeQES) for conducting surveys. Using this workflow, researchers are required to preregister their study before data collection, i. e., make available hypotheses, study design and planned analyses. The preregistration is presented to both the scientific community and an ethics board, which needs to approve of the planned form of the study. If approval is given, the study specification is uploaded to PeQES, which autonomously collects survey responses from study participants and performs statistical analyses on these data. Thereafter, the researcher is provided with the results of the analyses. The collected survey responses never leave the trusted platform. This approach ensures scientific integrity without the need to publish datasets of potentially personally identifiable information.

PeQES relies on *Trusted Execution Environments* (TEEs) for ensuring the confidentiality of survey responses and the integrity of the statistical analyses. However, this requires trusting the respective hardware’s manufacturer. Further, TEEs cannot offer absolute security. Some TEEs provide no protection against specific classes of attack (e. g., *Intel Software Guard Extensions* (SGX) does not protect against side-channel-based attacks [34]). Additionally, TEEs can (and regularly do) contain security vulnerabilities [115, 26, 74, 125, 28].

For this reason, this work explores a different approach for implementing a secure platform through which quantitative empirical studies can be conducted in a privacy-friendly manner. The proposed platform uses both TEEs and *Secure*

<sup>1</sup> E. g., 87% of the United States’ population are uniquely identifiable through a combination of ZIP-code, gender and date of birth [111]; 99.98% are identifiable through a combination of 15 demographic attributes [97].



*Multiparty Computation* (SMPC) in order to provide strong guarantees regarding the confidentiality of survey responses. We evaluate this approach with regards to security and efficiency. In particular, we answer the following research questions:

**RQ1:** What are the security guarantees provided by the proposed concept?

**RQ2:** How efficient is the proposed concept compared to baseline implementations using only hardware enclaves/only SMPC to secure the platform?

**RQ3:** Which existing technologies are suitable for implementing the proposed concept?

The remainder of this work is structured as follows. Chapter 2 introduces the mathematical concepts of secret sharing, SMPC, and TEEs. In Chapter 3, we introduce technologies that enable usage of these concepts in practice. Chapter 4 discusses related work, in particular attempts at strengthening privacy in empirical studies or data analysis in general through use of either TEEs or SMPC. Our approach for achieving the same goal through a combination of TEEs and SMPC is described in Chapter 5. In order to evaluate our approach in terms of computational performance, we implement a prototype. Our choice of technology and implementation details of this prototype are discussed in Chapter 6. The experimental apparatus and results of the evaluation are presented in Chapter 7. We discuss these results, their implications and limitations as well as possible future work in Chapter 8. We conclude our work in Chapter 9.



In this Section, we introduce relevant mathematical and technological concepts. Specifically, we describe secret sharing (Section 2.1), SMPC (Section 2.2) and TEEs (Section 2.3).

### 2.1 SECRET SHARING

*Secret sharing* describes the act of splitting some data  $D$  into  $n$  pieces  $D_1, \dots, D_n$  such that reconstruction of  $D$  is only possible with knowledge of  $k$  or more of these pieces. Any combination of  $k - 1$  or less pieces yields no information about  $D$  at all. A scheme describing such operations is called a  $(n, k)$  secret sharing scheme.

$(n, 1)$  secret sharing schemes and  $(n, n)$  secret sharing schemes can be constructed trivially. A  $(n, 1)$  secret sharing scheme is accomplished by setting  $D = D_1 = D_2 = \dots = D_n$ ; a  $(n, n)$  secret sharing scheme is realised by selecting randomly  $D_1, \dots, D_n$  such that  $\sum_{i=1}^n D_i = D$ . A more general  $(n, k)$  secret sharing scheme is proposed by Shamir [104] and discussed in Subsection 2.1.1.

#### 2.1.1 SHAMIR'S SECRET SHARING

Shamir's secret sharing [104] is a technique for constructing  $(n, k)$  secret sharing schemes for any  $1 \leq k \leq n$ . It is based on the fact that any polynomial  $q(x)$  with  $\deg(q) = k - 1$  is unambiguously defined by  $k$  points  $(x_1, q(x_1)), (x_2, q(x_2)), \dots, (x_k, q(x_k))$  when  $x_i \neq x_j \forall i \neq j$ .

Consider a secret date  $D$ . We aim to share  $D$  among  $n$  parties  $P_1, \dots, P_n$  such that any set of no less than  $k$  parties can reconstruct  $D$ . In order to accomplish this, Shamir proposes the following algorithm: First, choose a large prime  $p > D$ . The value of  $p$  can (and must) be made public; all of the following calculations are performed over the finite field  $\mathbb{Z}_p$ . We then construct a polynomial of degree  $k - 1$ , which we call  $q(x)$ . As a polynomial,  $q(x) = a_0 + a_1x + \dots + a_{k-1}x^{k-1}$ . We choose  $a_0 := D$ ; the remaining coefficients  $a_1, \dots, a_{k-1}$  are randomly chosen from a uniform distribution over  $\mathbb{Z}_p$ .

$D$  is then split into  $n$  shares by determining  $n$  distinct points located on the polynomial. More precisely, the shares are given by  $D_i = (i, q(i))$ ,  $i \in \{1, \dots, n\}$ . These shares are now distributed among the parties, i. e.,  $P_i$  receives  $(i, q(i)) \forall i \in \{1, \dots, n\}$ .

Recovering  $D$  is achieved by reconstructing  $q$  through interpolation and evaluating  $q(0)$ . Because  $q$  has degree  $k - 1$ , this process requires the secret shares of at least  $k$  participants. Consider the case of a group of  $k$  participants  $P_{i_1}, \dots, P_{i_k}$  that want to collaborate in order to restore  $D$  from their secret shares  $D_{i_j} = (i_j, q(i_j))$ ,  $j \in \{1, \dots, k\}$ . The group can calculate  $q$ 's coefficients  $a_0, \dots, a_{k-1}$  by forming a set of  $k$  linear equations:

$$\begin{aligned}
 a_0 + a_1 \cdot i_1 + \cdots + a_{k-1} \cdot i_1 &= q(i_1) \\
 a_0 + a_1 \cdot i_2 + \cdots + a_{k-1} \cdot i_2 &= q(i_2) \\
 &\vdots \\
 a_0 + a_1 \cdot i_k + \cdots + a_{k-1} \cdot i_k &= q(i_k)
 \end{aligned}$$

This system can be solved by performing the well-known *Gaussian Elimination* algorithm [38]. Because the matrix representation of this system is a Vandermonde matrix and  $i_j \neq i_h \forall j \neq h$ , its rows are linearly independent [110]. Hence, the system has a unique solution for  $a_0 = q(0) = D$ .

In practice, interpolation is typically done via Lagrange interpolation, i. e., through Equation 2.1.

$$q(x) = \sum_{s=1}^k q(i_s) \prod_{\substack{1 \leq j \leq k \\ j \neq s}} \frac{x - i_j}{i_s - i_j} \tag{2.1}$$

If we evaluate  $q$  at  $x = 0$  (as  $D = q(0)$ ), we arrive at Equation 2.2, which is an explicit formula for  $D$  [38, 110].

$$D = q(0) = \sum_{s=1}^k q(i_s) \prod_{\substack{1 \leq j \leq k \\ j \neq s}} \frac{i_j}{i_j - i_s} \tag{2.2}$$

**Example** Suppose we have a secret value  $D = 8080$  which we would like to share amongst four parties  $P_1, \dots, P_4$  such that any three parties can recover  $D$ . We do this via a  $(4, 3)$  Shamir's secret sharing scheme using  $p = 19\,991$  (i. e., all calculations are performed in  $\mathbb{Z}_{19\,991}$ ).

**Splitting  $D$**  We chose a random polynomial  $q$  of degree  $k - 1 = 2$  by setting  $a_0 = D = 8080$  and selecting coefficients  $a_1 = 4638$  and  $a_2 = 17\,126$  randomly from a uniform distribution over  $\mathbb{Z}_{19\,991}$ . Four secret shares  $D_1, \dots, D_4$  are then given by  $(1, 9853), (2, 5896), (3, 16\,200), (4, 783)$ .

These shares are then distributed among the parties such that (only)  $P_i$  gains knowledge of  $D_i$  for  $i \in \{1, \dots, 4\}$ .

**Recovering  $D$**  Any set of parties  $S \subseteq \{D_1, D_2, D_3, D_4\}$  where  $|S| \geq 3$  can combine their secret shares to gain knowledge of  $D$ . Suppose that  $S = D_1, D_3, D_4$ . We define  $i_1 = 1, i_2 = 3, i_3 = 4$  and recover  $D$  by applying Equation 2.2:

$$\begin{aligned}
 D &= \sum_{s=1}^k q(i_s) \prod_{\substack{1 \leq j \leq k \\ j \neq s}} \frac{i_j}{i_j - i_s} \\
 &= q(i_1) \frac{i_2}{(i_2 - i_1)} \frac{i_3}{(i_3 - i_1)} + q(i_2) \frac{i_1}{(i_1 - i_2)} \frac{i_3}{(i_3 - i_2)} \\
 &\quad + q(i_3) \frac{i_1}{(i_1 - i_3)} \frac{i_2}{(i_2 - i_3)} \\
 &= q(1) \frac{3}{(3-1)} \frac{4}{(4-1)} + q(3) \frac{1}{(1-3)} \frac{4}{(4-3)} + q(4) \frac{1}{(1-4)} \frac{3}{(3-4)} \\
 &= 9853 \cdot 2 + 16200 \cdot (-2) + 783 \cdot 1 \\
 &= 8080
 \end{aligned}$$

## 2.2 SECURE MULTIPARTY COMPUTATION

SMPC is a cryptographic primitive that allows a number of connected parties to evaluate a function in a secure manner [76]. The problem SMPC solves is the following one [122]: A group of  $n$  parties wishes to jointly evaluate a function. Each party is in possession of one secret input. Can the group evaluate the function in such a way that no party must reveal their input (or any information about their input) to other parties?

It turns out that every computable function can be evaluated as described above [46]. Secure computation is even possible if some parties collude in order to reveal information about an *honest* party's input or to change the result of the computation. In this case we say that the colluding parties are under the control of an adversary and call these parties *corrupted*.

In the following Section, we give an overview about the terminology used to describe protocols for SMPC, describe different attacker models and give an example of a protocol for general SMPC.

### 2.2.1 EXAMPLE: A SIMPLE SMPC PROTOCOL

In order to illustrate SMPC, we give a practical example of a protocol that can be used to perform a real-life multiparty computation: Consider a group of  $n$  employees ( $P_1, \dots, P_n$ ) that want to calculate the mean of their incomes. As workplace culture forbids revealing one's salary to others, the individual employees' incomes ( $s_1, s_2, \dots, s_n$ ) must remain private. In order to still be able to compute the average income, the employees execute the following protocol: Employee  $P_1$  chooses a random number  $r$ . They compute  $m_1 = r + \frac{s_1}{n}$  and send  $m_1$  to  $P_2$ . Employee  $P_2$  computes  $m_2 = m_1 + \frac{s_2}{n}$  and sends  $m_2$  to  $P_3$ . Employee  $P_3$  calculates  $m_3 = m_2 + \frac{s_3}{n}$  and sends it to  $P_4$ . Essentially, each employee adds their salary (divided by  $n$ ) to the message they receive and sends this sum to the next employee. Eventually, employee  $P_n$  sends  $m_n (= r + \sum_{i=1}^n \frac{s_i}{n})$  to  $P_1$ . Since  $P_1$  knows  $r$ , they can easily calculate  $m_n - r = \frac{1}{n} \sum_{i=1}^n s_i$ , which is the mean salary.  $P_1$  must then reveal this result to the other employees, completing the multiparty computation [33, 24].

Given such a protocol, one might ask whether or not this procedure can be considered secure. Indeed, the protocol guarantees the confidentiality of the parties' inputs in the presence of at most one corrupted party. However, no such guarantee exists against an adversary controlling two parties. Consider the case where party  $P_i$  and  $P_{i+2}$  are corrupted. Together, parties know the values  $m_i$  and  $m_{i+1}$ . Since  $m_{i+1} = m_i + \frac{s_{i+1}}{n}$ , party  $P_{i+1}$ 's salary  $s_{i+1}$  can be computed via  $s_{i+1} = n \cdot (m_{i+1} - m_i)$ . Further, party  $P_1$  can trivially perform a *denial of service* attack by not publishing the result of the computation. The protocol ensures the participants' privacy only against an adversary controlling a single party, but is susceptible to denial-of-service attacks by such an attacker. We therefore say that the protocol is not secure against more than one corrupted parties and defer the definition of this adjective to Subsection 2.2.2. In Subsection 2.2.4, we introduce SMPC protocols that are more general and provide much stronger security guarantees than the example protocol introduced above.

### 2.2.2 SECURITY

In order to analyse and compare protocols for SMPC, a precise definition of the term *security* is needed. The standard definition as of today [27] is called the *ideal/real simulation paradigm*. Consider an ideal world, in which an incorruptible *Trusted Third Party* (TTP) exists. In order to perform an SMPC of some function, the parties send their inputs to this TTP. The TTP computes the result of the function locally and provides each party with its output. As the individual parties never communicate with each other and the TTP is incorruptible, an adversary can learn no more than the inputs and outputs of the corrupted parties. It cannot modify the computation's results (apart from modifying the corrupted parties' inputs before computation starts) and cannot perform a denial of service attack.

We now define a protocol for multiparty computation as being secure if executing it amounts to emulating the ideal world. A protocol is said to emulate the ideal world if the following condition is true: for every output distribution that can be achieved by any adversary in the protocol under test there exists an adversary in the ideal model that can perform actions to achieve (essentially) the same output distribution. In other words: a secure protocol only permits the attacker to perform attacks that are also possible in the ideal model. Since no attacks are possible in the ideal model, neither are attacks on a secure real-world protocol [27, 76].

This notion of security encompasses the following properties [76]:

1. **Privacy:** No party learns more than its output. Depending on the computation at hand, information on the other parties' inputs might be deduced from the output. For example, an SMPC between two parties calculating the sum of their inputs inevitably enables the parties to calculate the other party's input.
2. **Correctness:** The result each party receives is guaranteed to be correct. Adversaries cannot influence the result of computations in a different way than to modify their controlled parties' inputs.
3. **Independence of Inputs:** The inputs of corrupted parties are chosen independently from the inputs of honest parties. For example, take the case

of an online auction using SMPC to determine the highest bid. Given independence of inputs, an attacker cannot generate a bid that is higher than the other parties' without knowing their bids.

4. **Guaranteed Output Delivery:** Corrupted parties cannot perform denial of service attacks and thus prevent honest parties from receiving their outputs.
5. **Fairness:** Either every party receives their output, or no party does. This property is implied by property 4, however, fairness does not imply property 4.

These five properties can be seen as (non-sufficient) requirements for a protocol to be called secure. In some applications, however, the requirements of fairness and guaranteed output delivery are dropped. This can either be because constructing a fair protocol is impossible for the application in question (e. g., for coin tossing [32]) or because requiring fairness or guaranteed output delivery would diminish performance. Protocols that omit these two requirements are called *secure with abort* [76].

### 2.2.3 POSSIBLE ATTACKER MODELS

In order to analyse a protocol for multiparty computation regarding its security, one must define what actions the adversary is capable of. This definition is then called an *attacker model*. Such a model has two main aspects: First, an attacker has a *corruption strategy*, which describes at what points in the computation parties can change their status from honest to corrupted and vice versa. The second aspect concerns *allowed adversarial behaviour*, i. e., what actions the corrupted parties can perform. We explain possible corruption strategies and possible definitions for allowed adversarial behaviour in the remainder of this Section.

#### **Corruption Strategy**

An attacker's corruption strategy defines at which point in time parties can switch from being honest to being corrupted or from being corrupted to being honest [76]:

1. *Static corruption:* The set of corrupted parties is determined and fixed before the computation begins. Parties cannot change from honest to corrupted or from corrupted to honest during the computation.
2. *Adaptive corruption:* The adversary can corrupt parties during the computation at will. It can adapt its decision which parties to corrupt based on the internal states of the already-corrupted parties, hence the name. Once corrupted, parties do not become honest again.
3. *Proactive security:* Parties can become corrupted throughout the computation, but may also switch from being corrupted to being honest. This models situations where corruption by an adversary can be reversed (e. g., by resetting an infiltrated machine to a clean state).

### Allowed Adversarial Behaviour

The allowed adversarial behaviour defines what actions corrupted parties can take. We distinguish between the following types of attacker [76]:

1. *Semi-honest* attackers (sometimes also called *honest-but-curious* or *passive*) are the weakest form of attacker. They may read the internal state of corrupted parties as well as messages sent and received by them. They attempt to use this information to gain more information than contained in the computation's result, however, they cannot modify or forge messages.
2. *Malicious* (or *active*) adversaries can access the corrupted parties' internal states and instruct these parties to deviate arbitrarily from the protocol. Besides sending wrong or invalid information, corrupted parties might also forge or withhold messages.
3. *Covert* adversaries have the same capabilities as malicious adversaries. However, they do not wish to be caught acting maliciously, and hence do not perform actions that lead to their detection with some application-dependent probability [11].

#### 2.2.4 A MORE GENERAL SMPC PROTOCOL

In this Section, we introduce a protocol for SMPC using Shamir's secret sharing as a basic building block [43, p. 43-44], which is known as the *BGW* protocol [17]. We denote the number of parties participating in the computation with  $n$ . This protocol requires the function that the parties would like to compute to be represented as an arithmetic circuit over the finite field  $\mathbb{Z}_p$  ( $p > n$ ) comprised of addition, multiplication-by-public-constant and multiplication gates. We assume that this circuit is known to all parties. Further, we assume the existence of pairwise secure channels, such that all parties can communicate securely with each other. The protocol then consists of the following phases:

1. *Input sharing*: The parties share their input with the other parties. Each party splits its input into  $n$  secret shares and distributes them among the other parties. For this, a  $(n, t + 1)$  Shamir's secret sharing scheme where  $t = \lfloor \frac{n-1}{2} \rfloor$  is used. Party  $P_i$ 's secret input value  $D_i$  is split into the shares  $D_{i,1}, \dots, D_{i,n}$ . It distributes these shares such that party  $P_j$  is provided with  $D_{i,j}$  over a secure channel. After this step, the parties are in possession of  $n$  secret shares, each of which stems from a different party and thus corresponds to a different input wire in the circuit.
2. *Circuit evaluation*: The parties evaluate the circuit gate by gate and exchange messages as needed. We now describe the steps necessary to evaluate the different types of gate [43, 17, 76].

**Addition Gate** The polynomials  $a(x)$  and  $b(x)$  represent secret-shared values on the addition gate's input wires. Party  $P_i$  possesses values  $a(i)$  and  $b(i)$ . The output wire of an addition gate should hold  $a(0) + b(0)$ , i. e., each party should have valid secret share of  $a(0) + b(0)$  after evaluation of the gate. For that,  $P_i$  can simply compute  $a(i) + b(i)$ , thereby acquiring



value  $c(i)$  of polynomial  $c(x) = a(x) + b(x)$ . Since  $c(0) = a(0) + b(0)$  and  $\deg(c) = t$ , the value  $c(i)$  is a valid secret share of  $a(0) + b(0)$ .

**Multiplication-by-public-constant Gate** Given a public constant  $h$  in plaintext and a secret share of the gate's input value (the corresponding polynomial  $a(x)$  evaluated at point  $i$ , i. e.,  $a(i)$ ), party  $P_i$  must compute  $h \cdot a(i)$ . We define  $c(x) = h \cdot a(x)$  and, since  $\deg(c) = t$ , immediately see that  $c(i) = h \cdot a(i)$  is a valid secret share of  $c(0) = h \cdot a(0)$ .

**Multiplication Gate** Consider a multiplication gate. The gate's input wires hold values encoded by the polynomials  $a(x)$  and  $b(x)$ . Each party possesses one share of these two values, i. e.,  $P_i$  has knowledge of  $a(i)$  and  $b(i)$ . The goal is to have the output wire hold the value  $a(0) \cdot b(0)$  encoded by a polynomial with degree  $t$ , with party  $P_i$  holding one share. As for the previous gates, parties can multiply their shares locally, computing  $c(x) = a(x) \cdot b(x)$ . While  $c(0) = a(0) \cdot b(0)$ , we observe that  $\deg(c) = 2t$ , which means that  $c$  is not suitable for use in a  $(n, t + 1)$  secret sharing scheme. In order to obtain a polynomial representation of  $c(0)$  that has degree  $t$ , the parties perform a degree-reduction step. For this, each party  $P_i$  selects a uniformly distributed random value  $r_i$ . It splits this value into shares in two different ways: First, using a  $(n, t + 1)$  secret sharing scheme (using a polynomial  $R_t^i(x)$  of degree  $t$ ) and second using a  $(n, 2t + 1)$  secret sharing scheme (using a degree  $2t$  polynomial  $R_{2t}^i(x)$ ). These shares are then distributed among the parties such that party  $P_i$  ends up with shares  $R_t^i(i)$  and  $R_{2t}^i(i) \forall j \in \{1, \dots, n\}$ . Each party then computes  $R_t(i) = \sum_{j=1}^n R_t^j(i)$  and  $R_{2t}(i) = \sum_{j=1}^n R_{2t}^j(i)$ . The polynomials  $R_t(x)$  and  $R_{2t}(x)$  are thus two independent sharings of  $r = \sum_{j=1}^n r_j$  (i. e.,  $R_t(0) = R_{2t}(0) = r$ ) and party  $P_i$  holds one share of each of these sharings. Further, it is evident that no party knows the value of  $r$ .

Each party  $P_i$  locally computes a share of the polynomial  $d(x) = c(x) - R_{2t}(x)$  by evaluating  $d(i) = c(i) - R_{2t}^i(i)$ . Note that  $\deg(c) = \deg(R_{2t}) = \deg(d) = 2t$ . The parties broadcast their shares of  $d(x)$  and reconstruct  $d(0) = c(0) - r$ . Party  $P_i$  can then calculate  $c'(i) = R_t(i) + d(0)$ . The polynomial  $c'(x)$  with  $\deg(c') = t$  then encodes the desired result of the multiplication since  $c'(0) = R_t(0) + d(0) = r + c(0) - r = a(0) \cdot b(0)$ . Thus, each party holds a secret share of the input wires' product encoded in a degree- $t$  polynomial. Further, the protocol the parties execute to acquire  $c'(i)$  does not give them any information about the values  $c(0)$ ,  $c'(0)$ ,  $a(0)$  or  $b(0)$  [76, 37].

3. *Output reconstruction:* After evaluating the circuit, each party holds one share of the value on each output wire. In order to reconstruct these values, the parties exchange their shares. An output value can be reconstructed through interpolation (see Subsection 2.1.1) after  $k$  of its shares have been obtained.

This protocol is secure in a scenario in which every pair of parties can communicate privately, the attackers are semi-honest and a majority of the parties (i. e.,  $> \frac{n}{2}$ ) is not corrupted. This is because when sharing the inputs

in Step 1, the parties use a  $(n, t + 1)$  secret sharing scheme where  $t = \lfloor \frac{n-1}{2} \rfloor$ , implying that a semi-honest coalition of  $\lfloor \frac{n}{2} \rfloor$  is sufficient to reconstruct the inputs of all parties. Further, it is secure against a coalition of  $< \frac{n}{3}$  active attackers, if a *verifiable* secret sharing scheme (e. g., the one proposed by [31]) is used instead of Shamir’s secret sharing [10].

We note that protocols tolerating up to  $\frac{n}{2}$  malicious parties are known [15, 92], however, these protocols assume the existence of a broadcast channel the parties have access to.

### 2.2.5 PERFORMANCE IMPROVEMENTS THROUGH PREPROCESSING

In Subsection 2.2.4, we introduce a protocol for general-purpose SMPC in an honest-majority scenario with semi-honest attackers. However, the protocol requires communication at various points in time throughout execution. First, input sharing and output reconstruction require communication, which cannot be avoided in SMPC protocols based on secret sharing techniques. Additionally, each evaluation of a multiplication gate requires two rounds of communication: one for cooperatively establishing  $R_t(x)$  and  $R_{2t}(x)$ , and one for reconstructing  $d(0)$ . This poses serious limitations on the performance and scalability of the protocol.

For this reason, we give an example of a modification of the above protocol that reduces interactivity through preprocessing. This modification was first proposed by Beaver [14] and has found use in several real-world protocols used for SMPC. It makes use of *Beaver triplets*, which are tuples of three numbers  $(a, b, c) \in \mathbb{Z}_p^3$  such that  $ab = c$ .

The protocol introduces a preprocessing phase, which is executed before the above protocol commences. We denote the number of multiplication gates in the arithmetic circuit to be evaluated with  $N$ . During preprocessing, the parties generate  $N$  Beaver triplets  $(a_k, b_k, c_k) \forall k \in \{1, \dots, N\}$  such that after preprocessing, each party  $P_i$  holds one share of the numbers in the triplets. In order to generate one Beaver triplet  $(a_k, b_k, c_k)$ , each party  $P_i$  chooses two random numbers  $a_{k,i}$  and  $b_{k,i}$ , shares them using a  $(n, t + 1)$  secret sharing scheme (producing shares  $a_{k,i,j}$  and  $b_{k,i,j} \forall j \in \{1, \dots, n\}$ ) and distributes these shares among the parties. After that,  $P_i$  is in possession of  $a_{k,\ell,i}$  and  $b_{k,\ell,i} \forall \ell \in \{1, \dots, n\}$  and computes its shares of  $a_k$  and  $b_k$  as  $\sum_{\ell=1}^n a_{k,\ell,i}$  and  $\sum_{\ell=1}^n b_{k,\ell,i}$ . Afterwards, the parties perform the protocol for evaluating a multiplication gate as described above with shared inputs  $a_k$  and  $b_k$  in order to acquire a share of  $c_k$ . This process is repeated until  $N$  Beaver triples have been generated [14].

These triplets can now be used to reduce interactivity in the circuit evaluation phase. For this, the evaluation protocol of multiplication gates is modified, which we describe in the following<sup>1</sup>. A Beaver triplet  $(a_k, b_k, c_k)$  is used to aid evaluation of multiplication gate  $g_k$ .

Suppose  $P_i$  wants to evaluate  $g_k$  with secret shared values  $s_1(0)$  and  $s_2(0)$  (encoded through polynomials  $s_1(x)$  and  $s_2(x)$  such that  $P_i$  knows  $s_1(i)$  and  $s_2(i)$ ) on  $g_k$ ’s input wires with the help of Beaver triplet  $(a_k, b_k, c_k)$ , which is

<sup>1</sup> This description is based on Beaver [14] and partly on Jiff’s source code, which implements this protocol faithfully. See: <https://github.com/multiparty/jiff/blob/8ea565d3d0becde8f71243fb9daea6ef0ba9bb7e/lib/client/protocols/numbers/arithmetic.js#L192-L218>

shared as polynomials  $(A_k(x), B_k(x), C_k(x))$ . It is in possession of the values  $s_1(i), s_2(i), A_k(i), B_k(i)$  and  $C_k(i)$ . First,  $P_i$  calculates  $d(i) = s_1(i) - A_k(i)$  and  $e(i) = s_2(i) - B_k(i)$ . The parties then cooperate in order to reconstruct  $d(0)$  and  $e(0)$  (which does not reveal any information about  $s_1(x)$  and  $s_2(x)$ ). Now, each party  $P_i$  can compute a share of the input wires' product  $s_3(i)$  through  $s_3(i) = d(0) \cdot e(0) + B_k(i) \cdot d(0) + A_k(i) \cdot e(0) + C_k(i)$ . This result is correct, because the polynomial  $s_3(x) = d(0) \cdot e(0) + B_k(x) \cdot d(0) + A_k(x) \cdot e(0) + C_k(x)$  has degree  $t$  and

$$\begin{aligned}
s_3(0) &= d(0) \cdot e(0) + B_k(0) \cdot d(0) + A_k(0) \cdot e(0) + C_k(0) \\
&= \left( s_1(0) - A_k(0) \right) \cdot \left( s_2(0) - B_k(0) \right) + B_k(0) \cdot \left( s_1(0) - A_k(0) \right) \\
&\quad + A_k(0) \cdot \left( s_2(0) - B_k(0) \right) + C_k(0) \\
&= s_1(0) \cdot s_2(0) - s_1(0) \cdot B_k(0) - s_2(0) \cdot A_k(0) + A_k(0) \cdot B_k(0) \\
&\quad + s_1(0) \cdot B_k(0) - A_k(0) \cdot B_k(0) + s_2(0) \cdot A_k(0) - A_k(0) \cdot B_k(0) \\
&\quad + A_k(0) \cdot B_k(0) \\
&= s_1(0) \cdot s_2(0)
\end{aligned}$$

Further, no information about  $s_3(0)$  is revealed, as the numbers comprising the utilised Beaver triplet are uniformly distributed and unknown to the parties.

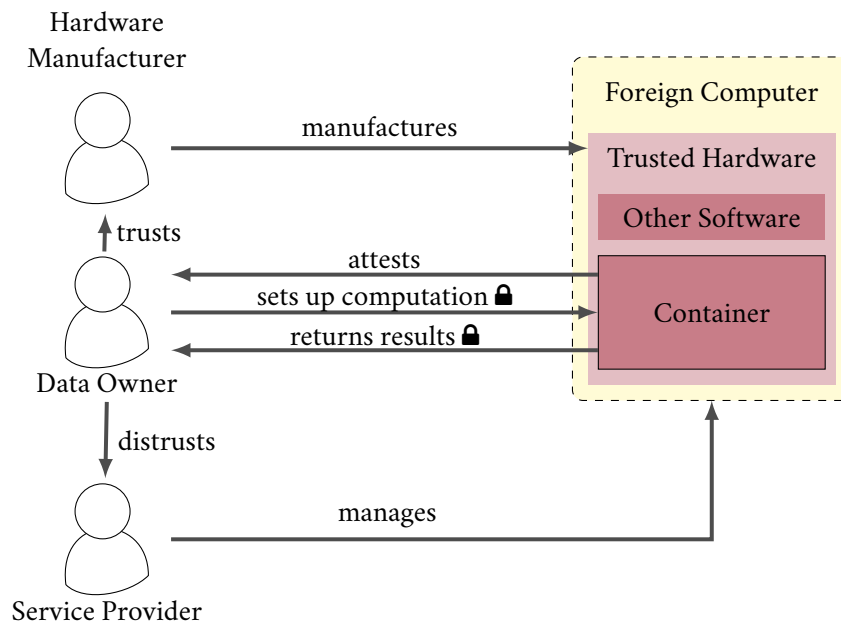
We note that evaluating a multiplication gate in this manner requires only one round of communication (when reconstructing  $d(0)$  and  $e(0)$ ). This performance improvement was bought at the cost of an additional preprocessing phase. However, preprocessing is independent of the parties' inputs to the computation, which means that it can be performed in advance before the inputs are available. In some use cases, this turns out to be a desirable property.

### 2.3 TRUSTED EXECUTION ENVIRONMENTS

It is common practice of businesses to perform computations not locally, but in the cloud. In particular, computations requiring powerful hardware or services requiring rapid scalability are often outsourced to specialised cloud providers, e. g., Amazon AWS, Microsoft Azure or Google Cloud. However, in the current data economy, code and especially data (particularly personal data) are highly valuable resources. Handing over these vital resources to cloud providers, which are often direct competitors, and possibly exposing these resources to other tenants sharing the same physical computing hardware conflicts business interests and data protection laws.

TEEs aim to alleviate these issues by providing secure *containers/enclaves*, which provide secure environments in which applications can be executed. These containers run on the cloud provider's machines, but are strictly isolated from all other applications running on the same hardware, in particular privileged software such as the operating system. This isolation is enforced by hardware. TEEs typically provide an *attestation* mechanism, through which the software running inside a container can prove to the user that it is indeed protected by trusted hardware and has not been tampered with.

A schematic overview of TEEs is shown in Figure 2.1. A user (which we also refer to as a *data owner*) owns private data; service providers own hardware



*Figure 2.1:* General architecture for trusted computing (based on Costan and Devadas [34]). A service provider manages computing infrastructure that is built on trusted hardware manufactured by a hardware manufacturer. A secure container hosted on trusted hardware attests to a data owner, i. e., proves its identity and that it is actually protected by trusted hardware. Through attestation, the data owner and the container additionally establish a secure communication channel. The data owner can then setup a computation by loading private data into the container. The container then executes the desired computation and securely returns the result to the data owner.

manufactured by a manufacturer. Data owners can trigger the creation of a secure container on the infrastructure owner’s hardware. After creation, they can perform an attestation to convince themselves that the container is indeed protected by trusted hardware and to establish a secure communication channel to the container. They can then upload their data, at which point computation inside the container commences. After the computation has finished, the results are transmitted to the user. Note that at no point in time, the service provider (or untrusted software running on the same hardware) has access to data, code or (intermediate) results of the computation running inside the container.

Examples of TEEs are

- Intel SGX [79, 55]
- ARM TrustZone [3]
- AMD Secure Encrypted Virtualization (SEV) [4]
- Execute-only Memory (XOM) [75]
- Trusted Platform Modules (TPM) [112]
- Sanctum [35]
- Keystone [73]

It is worth mentioning that ensuring the confidentiality of code and data in a cloud setting is not the only use case for TEEs. In general, TEEs protect software from other software and from the owner of hardware it is executed on. It thus finds (not uncontroversial [109, 7]) uses e. g., in digital rights management [124], authenticated boot mechanisms [45], hardening of sensitive applications [78] (e. g., programmes for banking), ensuring the integrity of mobile- and IoT-devices [64] and private contact discovery in messenger services [77].

In this work, we leverage the security guarantees of Intel SGX to implement a framework for secure statistical analysis of sensitive data. The following Section serves as an introduction into SGX, its mechanisms and the security it provides.

### 2.3.1 INTEL SGX

*Intel Software Guard Extensions* (Intel SGX) [79, 5, 55] is an implementation of a TEE available on a number of recent Intel CPUs. It was announced in 2013 and became available in 2015 with the introduction of Intel’s *SkyLake* architecture [6]. It has been part of recent incarnations of the Intel architecture, however, support on client platforms was dropped in 2022. Going forward, SGX will only be available on Intel Xeon processors [96].

SGX is comprised of hardware elements and a set of CPU instructions. These instructions allow processes to instantiate enclaves. Enclaves are isolated execution environments in the sense that their code and memory contents cannot be read or manipulated by hard- or software except by the enclave code itself and the Intel CPU hosting the enclave. Further, it provides the possibility to verify the integrity and identity of enclaves via attestation [113].

#### SGX Data Structures

Intel SGX introduces various data structures necessary for the secure operation of enclaves. In this Section, we describe the most essential data structures using the information provided in Intel’s *Software Developer Manual* [54] as well as information from McKeen et al. [79] and Costan and Devadas [34].

**Processor Reserved Memory** The *Processor-Reserved Memory* (PRM) is a contiguous part of the system’s RAM that is used for the *Enclave Page Cache*. It has hardware protections in place that prevent non-enclave read or write accesses, i. e., from system software, OS, hypervisor, system management mode code, and peripherals.

**Enclave Page Cache** The *Enclave Page Cache* (EPC) is used for storing the contents and management information of enclaves. It is divided into 4 kB pages and located in the PRM. The OS or hypervisor is responsible for assigning pages to enclaves using specialised SGX instructions. As these entities are not trusted in SGX’ attacker model, the assignment decisions of the OS/hypervisor are checked by the (trusted) processor and only performed if deemed security compliant. Because the EPC is located in the PRM, its pages can only be accessed by code running in enclaves. However, there is one exception to this rule: The SGX instructions used for the allocation of pages to enclaves also initialise these pages, possibly with data from non-PRM pages.

SGX implements a mechanism for page swapping, which can be used to securely evict EPC pages to untrusted RAM. For this, it provides the instruction EWB, which evicts a page to non-PRM. Because non-PRM is assumed to be malicious, EWB encrypts the page's contents using symmetric encryption and a nonce. This guarantees confidentiality, integrity and freshness. Once stored in non-PRM, the OS can treat evicted pages like regular memory pages and evict them further, e. g., to disk. The reverse operation, namely loading back evicted pages into the EPC, is implemented by the instructions ELDU and ELDB.

On older Intel processor architectures (i. e., architectures preceding *IceLake*), the EPC is limited to a size of 128 to 256 MB<sup>2</sup>. If an enclaved application allocates more memory, some of the enclave's pages are evicted into RAM, negatively impacting performance. More recent Intel processors do not have this limitation.

**Enclave Page Cache Map** The *Enclave Page Cache Map* (EPCM) stores the allocation status of pages, their type and an identifier of the enclave it has been allocated to. It is solely used by the processor to track the state of EPC pages and to validate the allocation decisions of the OS/hypervisor. The processor further uses it to enforce the isolation guarantees of Intel SGX, i. e., to ensure that enclaves only access memory that has been assigned to them. As a given page can only be assigned to one single enclave at a time, communication between enclaves using shared EPC memory is impossible.

**SGX Enclave Control Structure** Each enclave is associated with a *SGX Enclave Control Structure* (SECS), which is used to store metadata (e. g., enclave size, debug status etc.). SECSs are stored in the EPC, they are however managed by the processor and not accessible by enclaves themselves. This is necessary because the SECS stores the enclave's *measurement*, which is used for attestation.

## A Day in the Life of an SGX Enclave

In this Section, we describe the typical life-cycle (see Figure 2.2) of SGX enclaves [34].

**Creation and Initialisation** The creation of an enclave can be triggered by system software with the ECREATE instruction. This instruction allocates one EPC page, which is consequently used as the new enclave's SECS. This SECS is initialised with the contents of a non-EPC page owned by the application that issues the ECREATE instruction. It is required to contain valid values for all SECS fields. If this is not the case, the instruction results in a page fault or in a general protection fault. ECREATE additionally sets the new enclave's INIT attribute (which is stored in its SECS) to false. This marks the enclave as uninitialised and not yet ready to be scheduled for execution.

While the enclave is in this state, system software can perform the initialisation of the enclave. This process consists of allocating EPC pages to the enclave and initialising them using data from non-EPC pages. This is done using the EADD instruction, which accomplishes both of these tasks. Additionally, the system

<sup>2</sup> See: <https://gramine.readthedocs.io/en/latest/performance.html#choice-of-sgx-machine>

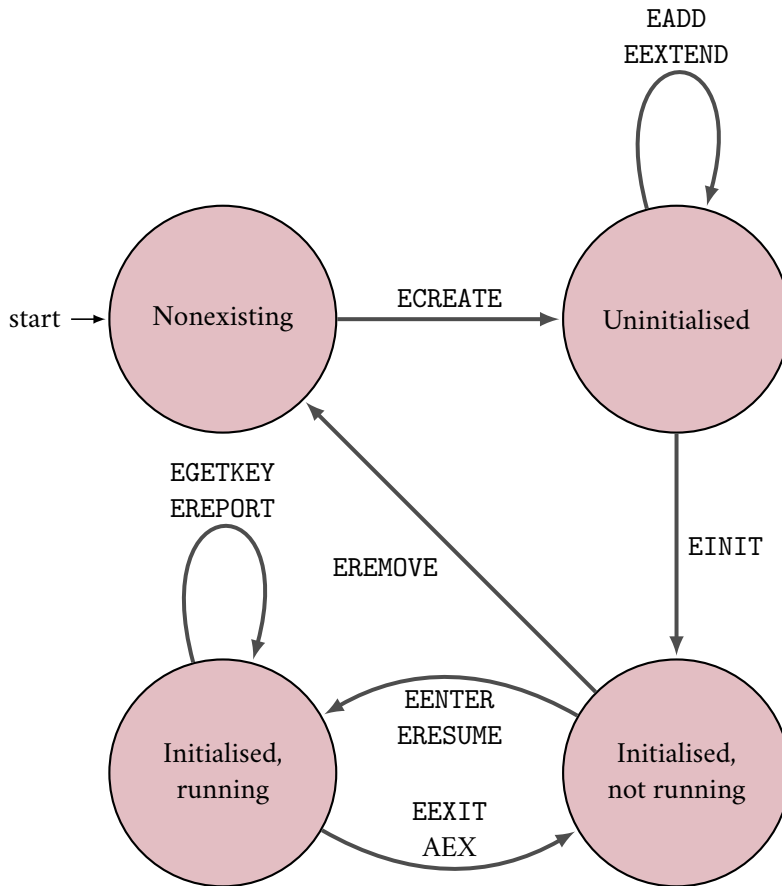


Figure 2.2: The life-cycle of an SGX enclave based on Costan and Devadas [34]. Some instructions available in SGX have been left out for brevity.

software uses the EEXTEND instruction for updating the enclave’s measurement after new pages have been added.

After the initial state of the enclave has been established, the system software can perform a procedure to mark the enclave as initialised and thus ready to launch. For this, it must obtain an EINIT token, which can only be issued by a special *Launch Enclave* (LE) provided by Intel. In order to be able to launch enclaves, one must thus first start the LE, which, due to its special nature, does not require an EINIT token. The LE inspects the newly created enclave and provides the system software with a MAC-protected EINIT token if it approves its launch. The criteria by which the LE decides whether or not to approve an enclave are not known. The implications of the existence of a LE and its role in the launch process are discussed in detail by Costan and Devadas [34, §5.9]. After having obtained an EINIT token, the system software can execute the instruction EINIT, which checks the validity of the EINIT token and marks the enclave ready for launch by setting the INIT attribute in the corresponding SECS to *true*. This causes subsequent invocations of EADD to fail, but allows (unprivileged) system software start to the execution of the enclave’s code.

**Entry and Exit** After an enclave has been initialised, an unprivileged<sup>3</sup> process (which we refer to as *host process*) can execute its code. For this, it invokes the SGX instruction EENTER. EENTER saves the process' context, sets the mode of the processor to *enclave mode* and jumps to a specified location in the enclave's code. Being in enclave mode allows the process to access the enclave's EPC, which is not possible in non-enclave mode.

The enclave code can use the instruction EEXIT to return to non-enclave code. This instruction sets the processor's mode to non-enclave, restores the host process' previously saved context and hands over control by jumping to an address in the host process' (non-EPC) memory space.

EENTER and EEXIT are used for *synchronous* entry and exit of enclave code, i. e., entry and exit during normal execution. However, as the execution of an enclave can be interrupted in unforeseen ways through hardware exceptions, an additional mechanism for keeping the enclave's contents safe throughout such events is needed. Intel SGX provides such a mechanism with a routine called *Asynchronous Exit* (AEX) and an instruction ERESUME. If a hardware exception occurs during the execution of enclave code, the processor performs an AEX. An AEX saves the enclave's execution context to the EPC and restores the host process' context previously saved by EENTER. It then invokes the system software's exception handler. After handling the hardware exception, the handler can execute ERESUME in order to resume execution of the enclave's code.

**Teardown** When an enclave is no longer needed, it can be torn down. For this, system software can execute the EREMOVE instruction on the EPC pages allocated to the enclave, freeing them. After all EPC pages belonging to the enclave have been freed, EREMOVE can be called on the EPC page holding the enclave's SECS, after which its removal is complete.

**Other Features** Intel SGX provides a range of additional features and specialised instructions. As the rest of this work makes no use of these features, we omit a detailed description, but only mention the existence and use of *enclave sealing* and *secret migration*, which are worth knowing.

Enclave sealing provides a mechanism to persist an enclave's memory contents in a secure way [5]. For this, SGX uses symmetric encryption with a key based on the enclave's identity. This ensures that only subsequent instantiations of the exact same enclave can decrypt the persisted state.

Secret migration provides a mechanism for migrating an enclave's secrets to a different enclave running a more recent version of the same software. A theoretical attack, in which an enclave could claim to be a newer version of an existing enclave in order to gain access to its secrets is mitigated by only allowing secret migration between enclaves of the same author.

### SGX Attestation

Attestation is a process by which an enclave can prove that it runs on trusted hardware, in a trusted container, with a trusted initial state. In SGX, an enclave's measurement is an essential part of attestation. The measurement of an enclave is computed during its creation using the EEXTEND instruction and can, for our

<sup>3</sup> In this context, *privileged* mainly refers to the operating system.



intents and purposes, be viewed as a secure hash over the initial contents of the enclave’s EPC pages. SGX provides two mechanisms for attestation. *Local attestation* can be used by an enclave to prove its identity to another enclave running on the same system. *Remote attestation* is used to prove the fact that the enclave is running on a system supporting a specified version of SGX using trusted hardware with specified initial memory contents to a remote entity.

**Local Attestation** In order to prove its identity (i. e., its measurement as well as the platform it is running on) to a *target enclave*, an enclave can execute the EREPORT instruction. This instruction produces a *report*, which binds a 64-byte enclave-supplied message to the enclave’s identity using a symmetric key that is only known to the target enclave and to the processor. The target enclave can verify this report using this symmetric key, which it can acquire using the EGETKEY instruction.

**Remote Attestation** Remote attestation relies on an enclave provided by Intel, the so-called *quoting enclave* and on an *attestation key*. The attestation key is provided by Intel during provisioning of the processor. It is encrypted using the *provisioning seal key*, which is based on a secret value stored in secure hardware (namely e-fuses), and can only be acquired by Intel-signed enclaves.

The process of remote attestation starts with a remote entity (*challenger*) requesting an enclave to authenticate itself. This *challenge* contains a nonce. The attesting enclave first attests locally to the quoting enclave, using the nonce provided with the challenge as the enclave-supplied message. As an Intel-signed enclave, the quoting enclave can acquire the provisioning seal key using EGETKEY, which it uses to decrypt the attestation key. The attestation key is then used to sign the local attestation report, which is forwarded to the challenger. The challenger can verify the validity of the report using an attestation verification service.

### Security Guarantees

Intel SGX aims to provide confidentiality, integrity and freshness guarantees to the contents of enclaves. In SGX’s view, all system software (except locally attested enclaves) and hardware (except the Intel CPU) are malicious, in particular the OS/hypervisor and the enclave’s host process. For this reason, system software cannot read or write an enclave’s EPC pages (except through EADD during enclave creation), which is enforced by hardware (however, enclave code can read and write non-EPC pages). Further, context switches that transition away from enclave code (e. g., as caused by a hardware exception) always lead to the erasure of the values the enclave stored in the processor’s registers, making the leakage of secrets through registers impossible. Freshness follows from the remote attestation mechanism described above, where the attesting enclave must include a challenger-provided nonce in its attestation report.

However, while SGX defends against attempts to directly read or write enclave memory, it does not defend against a range of indirect attacks. For example, SGX does not defend against attacks utilising the side-channels of power consumption, performance statistics (e. g., cache misses), branch statistics or page tables [53]. Defence against such attacks is left to enclave authors, who

can harden their application using known countermeasures, e. g., by relying on side-channel resistant cryptographic libraries<sup>4</sup> [25].

Since its introduction, the security of Intel SGX has often been challenged. A large body of work is dedicated to circumventing SGX' security guarantees and/or proposing protections against such circumventions. For the sake of brevity, we only point out that practical attacks on SGX have been (and continue to be) found and refer to Zheng et al. [126] for a short overview and to Fei et al. [44] for a detailed survey of such attacks.

---

<sup>4</sup> E. g., `intel-sgx-ssl` is a cryptographic library provided by Intel intended for usage in SGX applications: <https://github.com/intel/intel-sgx-ssl>

In Chapter 6, we implement a software prototype combining SMPC and Intel SGX. In this Section, we discuss software libraries and frameworks that promise to allow easy integration of these technologies.

### 3.1 ENABLING SECURE MULTIPARTY COMPUTATION

In this Section, we introduce three software libraries for performing SMPC. We chose these three libraries because they

- Do not limit usage through proprietary licensing schemes
- Permit computation between more than two parties
- Implement protocols based on Shamir’s secret sharing (the reason for this requirement is discussed in Chapter 5)
- Are focused on easy usability.

Other currently available libraries for SMPC are either restricted by proprietary licenses [20, 72], do not support computation between at least three parties [39, 95, 123, 98], are not based on Shamir’s secret sharing [121, 119, 120, 12, 108, 52], serve specific niches like machine learning [83, 71], or are retired [99]. Our choice of libraries we consider is based on work by Hastings et al. [50] and a list of software libraries for SMPC found on GitHub<sup>1</sup>.

#### 3.1.1 JIFF

*Jiff (JavaScript Implementation of Federated Functionality)* [84, 1, 2] is a software library providing primitives for general-purpose SMPC. As its name suggests, it is written in and for JavaScript. Jiff was developed in order to be used in (possibly pre-existing) web applications. As such, it has unique properties which distinguish it from other SMPC libraries. The first property follows directly from the fact that Jiff is written in JavaScript: Jiff-based applications are very portable. They can be run on desktop computers and mobile devices via a modern browser, but also on headless systems using JavaScript runtime environments such as *node.js*<sup>2</sup>. Second, Jiff aims to support dynamic SMPC, in which parties can join and leave during computations, and fault-tolerant SMPC in the sense that it can recover from crashes and network partitions. Further, Jiff is largely protocol-agnostic and can (in principle) be used with a wide range of different SMPC protocols. However, the current version of Jiff only implements one SMPC protocol. This protocol, which is based on BGW [17], supports primitives for performing various common computations on secret shares, e. g., addition, subtraction, multiplication, division, comparisons, bitwise operations etc.

---

<sup>1</sup> <https://github.com/rdragos/awesome-mpc#software>

<sup>2</sup> <https://nodejs.org/>

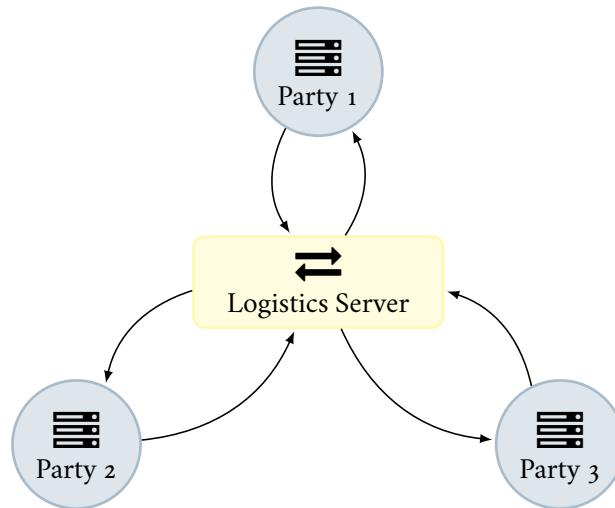


Figure 3.1: The general architecture of Jiff based on Albab [1]. The parties send messages to each other through the logistics server. As communication is end-to-end encrypted, the server learns nothing.

The general architecture of Jiff is shown in Figure 3.1. It consists of a number of parties and a *logistics server*, thus implementing a client-server architecture. The logistics server facilitates message transport between the parties and handles other organisational tasks such as announcing joining or leaving parties. Compared to the peer-to-peer architecture seen in most other software for SMPC, this has the very practical advantage that only the server needs a public, static network address (as opposed to every computing party). As all communication is encrypted, the server cannot learn anything about the computation itself. SMPC consists of a preprocessing phase and a computation phase. During preprocessing, the parties generate correlated randomness, e. g., Beaver triplets, which can be used to accelerate the subsequent computation phase (see Subsection 2.2.5). Parties can join or leave the SMPC while it is running. If they leave due to a network failure or crash, the server buffers messages sent to them and ensures that the computation can continue after they rejoin.

Further, the server has an interesting functionality for accelerating SMPC: it can act as a *crypto provider*, providing the parties with correlated randomness, e. g., Beaver triplets. In this case, the preprocessing phase is omitted and the parties can gather correlated randomness from the server as needed. Of course, using this functionality is a trade-off in security, which we discuss below.

The default SMPC protocol Jiff ships with enables an asymmetric security model [84]. The preprocessing phase uses a protocol based on BGW [17] (see Subsection 2.2.4) to generate Beaver triplets and is secure against a semi-honest minority. The protocol used in the computation phase, however, is secure against a semi-honest majority, i. e., up to  $n - 1$  corrupted parties cannot collude to reconstruct the secret input of the remaining party. This distinction is important, as Jiff allows an architecture in which the set of parties participating in preprocessing differs from the set of parties participating in the computation. If preprocessing is not used and Beaver triples are provided by the server, the security model changes. In this case, the computation remains secure against a majority of semi-honest non-server parties, but is insecure against a coalition of

the server and one or more non-server parties.

Jiff has support for *extensions*, which can introduce new functionality with minimal code changes. It ships with extensions providing support for arbitrarily large integers (*BigNumbers*), fixed-point numbers, negative numbers and alternative communication methods (e. g., HTTP long polling instead of the *socket.IO*<sup>3</sup> library, which is the default).

Unfortunately, Jiff’s documentation is incomplete. While the most frequently used interfaces are well-documented, documentation on internals, the provided protocol for SMPC and less-used functionality is often either outdated or non-existent.

### 3.1.2 MP-SPDZ

*MP-SPDZ* [61] is a framework implementing 30 different protocols for SMPC, which cover all common security models. Its aim is to provide a single interface to these protocols, enabling rapid comparison and benchmarking.

The framework consists of a compiler, which converts a high-level *Domain-Specific Language* (DSL) based on Python that implements business logic to bytecode, and a virtual machine that executes this bytecode.

In terms of efficiency, the framework performs relatively well. The reason for this is it applies various optimisations such as performing preprocessing on demand; it further enables programmers to implement message batching in a non-verbose way.

### 3.1.3 MPyC

*MPyC*<sup>4</sup> [101, 100] is a software framework for the Python programming language that implements SMPC based on Shamir’s secret sharing. It is secure against a semi-honest minority of corrupted parties. It has a strong focus on usability, i. e., it allows users to write relatively normal looking Python code which can either be executed locally or in a distributed fashion as an SMPC. Therefore, code using *MPyC* is usually more concise than code using other frameworks for SMPC, however this usability comes at a cost. First, *MPyC* typically performs computations slower than other frameworks [61]. Second, *MPyC* does not provide flexibility, neither in the choice of protocol used to perform the SMPC, nor in the system architecture. For example, implementing dynamic SMPC in which parties can join or leave the group of participants during execution is, technically speaking, possible, but requires circumvention of several abstractions *MPyC* provides and thus leads to unidiomatic code<sup>5</sup>.

## 3.2 ENABLING INTEL SGX

Intel SGX requires a specific programming model, in which applications are divided into non-enclave code and enclave code. In order to profit from SGX’

<sup>3</sup> <https://socket.io/>

<sup>4</sup> <https://www.win.tue.nl/~berry/mpyc/>

<sup>5</sup> See: <https://github.com/lschoe/mpyc/issues/42>

security guarantees, enclave code must fulfil certain properties. For example, it cannot trust the underlying OS, in particular data made available via the OS through system calls. For this reason, it is a widespread belief that porting an existing application for usage with SGX involves large code changes, which inhibits its adoption [113].

One approach for tackling this issue is the usage of a *Library OS*, which acts as a wrapper around the API provided by the system OS. It can thereby perform additional checks, e. g., verifying data provided by the OS, essentially shielding the application from the host OS. In 2015, it was shown that unmodified applications can be run in SGX with the help of a Library OS [13]. Using this approach, the entire application’s code is run inside an enclave. Entering and leaving the enclave (e. g., for serving system calls) is fully handled by the Library OS. While this tremendously increases the accessibility of SGX, using a Library OS comes with drawbacks. For example, as the entire application’s code is run in an enclave, all its system calls lead to an expensive enclave exit and reentry. In applications that are hand-written for SGX, only well-measured portions of the programme’s code are executed in an enclave, minimising the number of enclave entries and exits. Further, using Library OSes means including much code into the enclave, increasing the *Trusted Computing Base* (TCB). This does not only lead to a larger attack surface, but also increased enclave creation time [113].

Still, Library OSes have the potential to allow quick deployment of SGX-enabled applications without any need for refactoring. One incarnation of this approach is *Gramine-SGX*, which is discussed in the following Section.

### 3.2.1 GRAMINE-SGX

*Gramine*<sup>6</sup> [114, 113] (formerly called *Graphene*) is a library OS, which aims to isolate applications from other software running on a given system by re-implementing the Linux system call API. It provides platform adaptation layers for running applications in an isolated environment either with or without the help of Intel SGX enclaves. The latter mode, which we refer to as *Gramine-SGX*, allows users to run unmodified binaries in enclaves.

Gramine requires that applications come with a signed *manifest* file. This manifest file describes the resources the application is allowed to use, e. g., files that can be accessed or networking capabilities. In particular, each file that the application is allowed to use is associated with a secure hash of its contents. During runtime, Gramine enforces these rules using its implementation of the Linux system call API. For example, if an application requests to open a file using the open system call, Gramine first verifies that the target file’s hash equals the hash specified in the signed manifest. This ensures that the OS cannot inject malicious data into the enclave.

Gramine-SGX defends against an attacker model that is close to SGX’: it distrusts everything except the CPU and the own enclave’s contents. In particular, the OS, hypervisor, other (possibly privileged) applications and hardware outside the CPU package are viewed as potentially malicious.

Tsai, Porter and Vij [113] measure the overhead introduced by using Gramine or Gramine-SGX for a variety of applications. Compared to a programme run-

<sup>6</sup> <https://gramineproject.io/>

ning without techniques for isolation, Gramine (without SGX) introduces only a marginal overhead. Using Gramine-SGX leads to more overhead, increasing runtime by an order of magnitude. However, much of the overhead introduced by Gramine-SGX is due to enclave creation time, which takes several seconds. Without enclave creation time, the runtime overhead of Gramine-SGX is roughly a factor of 2-5.

### 3.2.2 SCRIPTSHIELD

*ScriptShield* by Wang et al. [118] is a framework that enables the execution of unmodified scripts in SGX enclaves. It accomplishes this by running an interpreter for the scripting language in question in an enclave. In order to enable system calls, they statically link the interpreter against a modified version of `libc`, which, much like a Library OS, verifies return values of system calls provided by the OS.

ScriptShield differs from Gramine-SGX in a number of ways. First, it relies on a remote party, which is expected to partake in attestation. After the enclave running ScriptShield has proven its authenticity, the remote party is expected to send a script to be executed through an encrypted channel. As far as we are aware, executing a script that is stored in untrusted storage is not supported. Second, ScriptShield has a smaller TCB than Gramine-SGX. This leads to smaller enclave sizes and faster startup times. Third, the authors' benchmarks suggest that ScriptShield has less runtime overhead than Gramine-SGX. However, these benchmarks are not directly comparable to the benchmarks performed by the authors of Gramine-SGX [113], as the former assess performance by executing standard benchmarking scripts, whereas the latter assess performance by measuring the runtime of real-world applications.

Currently, ScriptShield supports executing scripts written in either JavaScript, Lua or Squirrel. In principle, ScriptShield can be extended to support additional language interpreters, however, the authors note that this process, which involves statically linking the interpreter in question against a modified `musl-libc`<sup>7</sup>, requires "*tedious engineering efforts*" [118].

ScriptShield does not seem to be actively maintained. As of March 2023, the latest commit in the project's GitHub repository<sup>8</sup> is four years old. Further, we were unable to compile ScriptShield<sup>9</sup>.

<sup>7</sup> <https://musl.libc.org/>

<sup>8</sup> <https://github.com/OSUSecLab/scriptshield>

<sup>9</sup> See: <https://github.com/OSUSecLab/scriptshield/issues/1>





## RELATED WORK

---

In this Section, we discuss related work. Section 4.1 discusses attempts to improve the efficiency of SMPC by using TEEs as a TTP. Section 4.2 presents previous work focused on strengthening privacy guarantees in empirical studies or, more generally, in statistical data analysis.

### 4.1 IMPROVING SMPC PERFORMANCE THROUGH TEE

In today's world, network speed lags behind processor speed. Moore's law [49] describes the exponential growth in processing capabilities observed over the past decades. Nielsen [88] notes that a similar (albeit slower) growth can be observed for network bandwidth. However, latency in networks does not enjoy similar improvements. Part of the reason for this is the existence of propagation delay, which is the time needed by a signal to travel from its source to its destination. Unfortunately, as the speed of light is finite, new technological advances cannot decrease propagation delay meaningfully, placing a hard limit on network latency. For this reason, the performance of communication intensive applications is often bounded by the limitations of the network, not the processor [14].

SMPC is such an application, thus communication overhead significantly influences the performance of SMPC. For this reason, research has aimed to reduce communication requirements in SPMC settings using various cryptographic techniques [14, 16, 60]. Another interesting approach is to use TEEs to implement TTPs (as in the real/ideal simulation paradigm introduced in Subsection 2.2.2) [105, 91, 102, 48, 65, 9, 30]. These TTPs receive the parties' inputs, securely compute the results and send them to the parties. The parties can convince themselves that the computation is performed securely through attestation.

Portela et al. [91] implement SMPC using a TTP realised through Intel SGX. They report that their version of SMPC outperforms software-only SMPC by a factor of up to 300. These findings are consistent with Ankele and Simpson [8], who measure the computation times of simple operations using software-only SMPC and TEE-assisted SMPC and conclude that the latter outperforms the former by several orders of magnitude.

Gupta et al. [48] point out that using TEEs for implementing TTPs does not automatically make computations secure. First, trusting a TEE means trusting its designer, its manufacturer and its supply chain. The assumption that the TEE in use behaves as expected cannot be verified, because, to the best of Gupta et al.'s and our knowledge, techniques allowing customers to verify hardware are currently not widely available. Second, common TEEs (e. g., Intel SGX) provide no security guarantees against side-channel attacks. If software running in a container is not specifically written to resist such attacks, data may be leaked. Third, security depends on the assumption that the security guarantees claimed by the TEE are actually provided. However, vulnerabilities have been discovered in leading TEEs, calling into question the confidentiality and integrity actually guaranteed by these technologies [30].

Further, we note that some TEEs (one of which is Intel SGX<sup>1</sup>) only offer security in the presence of a computationally bounded attacker, whereas many protocols for SMPC that do not use TEEs provide unconditional security [76].

## 4.2 ENHANCING PRIVACY IN EMPIRICAL STUDIES

As discussed in Chapter 1, empirical studies are under threat of attacks that target the anonymity and privacy of their participants. Related work has aimed to alleviate this issue by hiding individual responses from researchers. Access is only granted to aggregations of the collected responses, which is assumed to make conclusions about individual respondents impossible.

In Subsection 4.2.1, we describe PeQES [81], which is a platform for conducting empirical studies. It hides the data of study participants in a TEE, making them inaccessible for researchers. In Subsection 4.2.2, we discuss approaches for privacy-respecting data analysis based on SMPC. None of the approaches we discuss are specifically designed as platforms for conducting empirical studies, but can in principle be modified to act as such.

### 4.2.1 PRIVACY THROUGH TEEs

Meißner et al. [81] propose a workflow for privacy-enhanced studies that relies on TEEs. In this workflow, researchers are required to specify a study design using a structured specification language, which includes the study procedure, the survey design and a script for statistical data analysis. Once this specification is approved and signed by an ethics board, researchers can upload it to a *platform for privacy-enhanced quantitative empirical studies* (PeQES). This platform conducts empirical data collection by allowing participants of the study to submit survey responses, e. g., through a web form generated according to the researchers' specification. Once data collection is completed (e. g., if a sufficient number of participants has submitted responses), PeQES runs the specified script for statistical data analysis and returns its results to the researchers.

The key ideas are that

- the ethics boards is trusted to only approve of scripts for statistical data analysis that do not have an unacceptable privacy impact for participants, i. e., results are aggregated in such a way that no personally identifiable information about participants is revealed
- PeQES only accepts study designs that have been signed by an ethics board
- PeQES protects the confidentiality of survey answers at all times using a suitable TEE technology.

In this workflow, researchers only have access to the results of the script for statistical data analyses. Interactive analysis or access to the responses themselves are no longer permitted, increasing privacy guarantees for participants. Meißner et al. provide a reference implementation of PeQES that uses Intel SGX as a TEE.

<sup>1</sup> This follows directly from the fact that Intel SGX uses computationally secure symmetric encryption (AES) for encrypting evicted EPC pages [34].

They show that the overhead introduced by SGX is acceptable even for studies with a large number of participants<sup>2</sup>.

#### 4.2.2 PRIVACY THROUGH SMPC

Lapets et al. [68, 69, 70, 66, 18] implement a platform to aggregate survey data in a privacy-preserving way, which consists of a service provider and a data analyser. Survey participants mask their responses using a randomly generated mask. Additionally, they encrypt the mask using the data analyser’s public key. Both the masked data and the encrypted mask are uploaded to the service provider, who computes an aggregation function on the masked data. Finally, the aggregated masked data as well as the encrypted masks are provided to the data analyser, who decrypts the masks with their private key, and uses them to obtain the unmasked aggregated data. In this system, privacy is guaranteed as long as the service provider and the data analyser remain semi-honest, i. e., do not collude. Lapets et al. give no comments on the performance of their platform, but their protocol suggests no major sources of overhead. However, their platform is not able to perform arbitrary computations. In the described form, the only possible aggregation is a sum over all responses, which is only useful in very specific use-cases. The platform has been used in a real-world scenario, in particular for analysing gender and ethnicity wage gaps in the Greater Boston Area. This analysis, which is performed on a yearly basis, later switched to using a platform [2] based on Jiff (see Subsection 3.1.1).

Subsequent work by partly the same authors proposes a conceptual framework for role-based SMPC, which is aimed at guiding the development of SMPC-based platforms as described above [67].

In general, we see that SMPC has been used in real-world deployments [41, 59, 21]. However, many of these deployments utilise specialised protocols that are unable to perform general purpose computation, but only specific functions. Such precisely tailored protocols often offer better performance than their general-purpose counterparts, but hinder extensibility and reusability. For this reason, we continue this Section with examples of related work developing platforms for data analysis which are not bound to one specific real-world use-case.

Jarrous and Pinkas [58] propose and implement *Canon-MPC*, which is similar to Jiff in that it supports participation in a computation via a web browser and enables asynchronous computation, in which parties do not need to be available throughout the computation. They use a non-interactive protocol for SMPC based on zero-knowledge proofs, which, as they claim, has excellent performance. However, Canon-MPC lacks auditability. Its source code is not accessible, it is delivered as a compiled binary and it is executed through Google’s proprietary *NativeClient*. Further, as *NativeClient* has been deprecated in June 2022<sup>3</sup>, we assume that Canon-MPC is no longer able to run in contemporary web browsers.

<sup>2</sup> The script for statistical data analysis in their sample study performs an independent two-sample t-test. In the case of 10 000 responses, SGX introduces an overhead of factor  $\approx 15$ , which is  $\approx 78$  s in absolute terms. Response submission times is independent of the number of participants or the script for data analysis and is not notably increased by usage of SGX.

<sup>3</sup> See: <https://blog.chromium.org/2020/08/changes-to-chrome-app-support-timeline.html>

## RELATED WORK

Chida et al. [29] implement a statistical analysis environment, where computations are performed in a distributed fashion using SMPC. This environment provides several pre-defined algorithms for statistical analysis, e. g., a version of the t-test and  $\chi^2$  test. Their implementation provides good performance and usability through an interface that allows specifying queries using the *R* language<sup>4</sup>.

Bogdanov et al. [23] design and implement a similar statistical analysis environment (*Rmind*). They provide a larger number of pre-defined algorithms. Further, they support restricting the analyses that can be performed on data via a study plan, which is a similar approach to protecting privacy as found in PeQES' [81] signed scripts for statistical analysis. They implement *Rmind* using the ShareMind [20] framework, however they do not publish the source code.

STAR [103] enables statistical tests on private data using a combination of SMPC, *Homomorphic Encryption* (HE) and append-only ledgers. In its envisioned workflow, data owners encrypt their data using HE. Researchers can then perform statistical tests on the encrypted data, yielding an encrypted result. Decryption of this result is only possible through an SMPC between a set of external computing servers. Further, STAR persists all operations that researchers perform in an append-only log file, which is assumed to be tamper-proof. STAR's primary purpose is to prevent questionable research practices by guaranteeing that such practices can be discovered by inspection of the log file. However, in this process, it also ensures the confidentiality of the utilised data. One limitation of STAR is that data must be provided by a single data owner, who performs encryption thereof. This makes it unsuitable for performing privacy-enhanced surveys without fully trusting the data owner, because responses are inherently accessible by it.

---

<sup>4</sup> <https://www.r-project.org/>

**Part II**

**Contribution**



## APPROACH

---

In this Section, we introduce our approach for providing strong security guarantees for empirical studies. Section 5.1 discusses the design of PeQES [81], which we introduced in Subsection 4.2.1, in detail. In Section 5.2, we introduce a platform similar to PeQES, but based on SMPC instead of TEEs. In Section 5.3, we introduce a combination of this platform and PeQES, which provides strong security guarantees regarding privacy in empirical studies.

### 5.1 USING SGX FOR HARDENING THE PRIVACY OF EMPIRICAL STUDIES

In this Section, we describe relevant parts of the design of PeQES as proposed by Meißner et al. [81]. PeQES relies on a central platform, which utilises a TEE (e. g., Intel SGX) for safeguarding participants' survey responses. In the following, we describe the protocol which the platform, researcher, ethics board and survey participants execute in order to securely conduct empirical studies.

First, we define the level of trust each entity enjoys. The platform, which we call  $TP$  and possesses a key pair  $(TP_{sk}, TP_{pk})$ , is assumed to be honest. This assumption follows from the fact that critical portions of the  $TP$  are protected by the TEE, which is assumed to be fully secure (i. e., no one except the platform itself has access to the data stored in protected memory).  $TP$  is hosted by a platform provider  $H$ , which acts as a covert adversary, i. e., tries to extract survey responses without being caught. Researcher  $R$  conducting the study is assumed to be fully malicious. The ethics board  $B$  (which possesses a key pair  $(B_{sk}, B_{pk})$ ) is honest, but not capable or willing to collect and process survey responses itself. Study participants  $P_1, \dots, P_m$  are assumed to be willing to participate in a given study, either motivated by a desire to contribute to science or because of other incentives, e. g., monetary rewards. They are not modelled as malicious, because we deem mitigation of attacks they are able to engage in to be out of scope for this work<sup>1</sup>. In order to conduct a study,  $R$  provides a study specification  $S$  that includes a survey and a script for statistical analysis. The survey can be in an arbitrary format understood by the participants, e. g., a HTML form. The script is assumed to be provided in a format both human- and machine readable, e. g., as a Python script.

The procedure for securely conducting a study is visualised in Figure 5.1.

- (1) Once  $R$  has finished developing  $S$ , they upload it to  $TP$ .
- (2)  $B$  performs a remote attestation with  $TP$ , verifying that the platform is secured by trusted hardware and establishing a shared symmetric key.
- (3)  $TP$  sends the study to  $B$  for approval.

<sup>1</sup> Study participants can perform Sybil attacks, which can be prevented by an authentication mechanism. Further, they can submit unwanted or invalid survey responses in order to acquire rewards, however, mitigation of such an attack is the responsibility of researcher, who can perform data cleaning as part of statistical analysis [81].

- (4) If  $B$  approves of the study, it signs it using its private key  $B_{sk}$ . It sends  $\text{Sig}_{B_{sk}}(S)$  to  $TP$ .
- (5) Participants download  $S$  and  $\text{Sig}_{B_{sk}}(S)$  from  $TP$  in order to participate. They are able to verify the signature using  $B_{pk}$ , which assures them that  $B$  has approved the study and attested  $TP$ , ensuring that it is running on trusted hardware.
- (6) After taking the survey, the participants send their encrypted responses to  $TP$ .
- (7) After enough responses have been recorded,  $TP$  runs the script for statistical data analysis contained in  $S$  and sends the results to  $R$ .

PeQES preserves the privacy of participants, because it guarantees that solely  $TP$  has access to the participants' raw responses. Researcher  $R$  is only provided with the results of the script for statistical data analysis, which, as ensured per  $B$ 's approval, do not contain personally identifiable information. Platform provider  $H$ , who controls the hardware  $TP$  is executed on, cannot access the raw responses because they are protected by a TEE.

Notably, the participants can be convinced that their data is protected by secure hardware without performing a remote attestation themselves. As the ethics board is trusted by the participants, its signature is sufficient to convince them that the integrity of the platform has been verified beforehand. Meißner et al. [81] describe this trust relationship with the term *transitive trust*.

The privacy guarantees of PeQES, however, only hold as long as the TEE used can provide adequate protection. If, e. g., Intel SGX is used, the platform remains vulnerable to a range of side-channel attacks, potentially exposing private information. For this reason, this work proposes an improved solution for ensuring the privacy of participants. Section 5.2 proposes a platform similar to PeQES, which relies on SMPC instead of TEEs for preventing unauthorised access to survey responses. Section 5.3 combines both platforms in order to ensure privacy in the presence of a large variety of attackers.

## 5.2 USING SMPC FOR HARDENING THE PRIVACY OF EMPIRICAL STUDIES

In the previous Section, we discuss PeQES, a system for conducting empirical studies that protects the privacy of participants. In this Section, we discuss a similar approach that, instead of a TEE, relies on SMPC for providing confidentiality. Essentially, it is based on the idea that participants split their responses into secret shares and distribute them among a set of servers. These servers perform statistical analysis on the responses via secret sharing based SMPC, however, they are unable to make conclusions about their contents.

The platform is no longer realised as a single entity, but as a distributed system consisting of  $n$  computation parties, which are denoted as  $C_1, \dots, C_n$  and have key pairs  $(C_{1,sk}, C_{1,pk}), \dots, (C_{n,sk}, C_{n,pk})$ . These parties are hosted by platform providers  $H_1, \dots, H_n$ . We assume that there is no set of more than  $k = \lfloor \frac{n-1}{2} \rfloor$  colluding platform providers, i. e., we assume a honest majority. We assume that the computation parties can communicate with each other in a secure way. Further, a semi-honest logistics server  $L$  facilitates communication



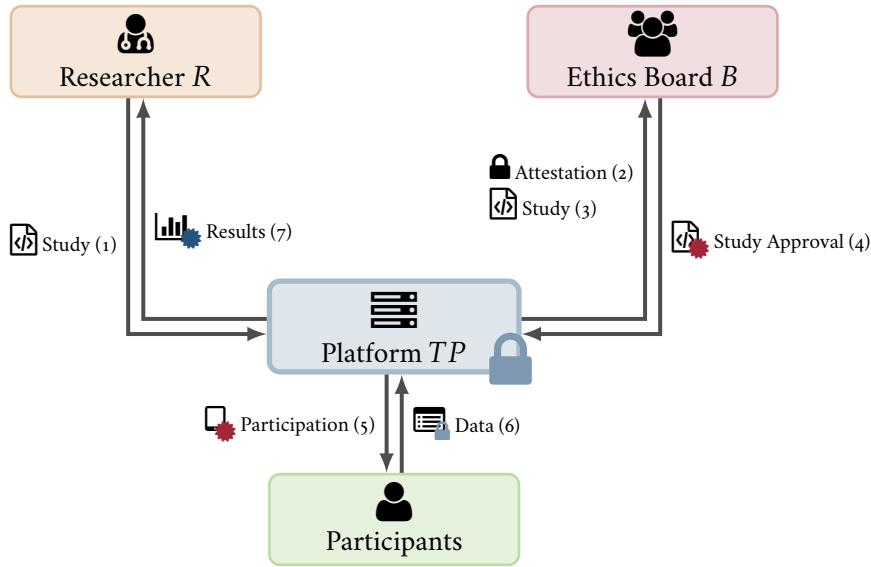


Figure 5.1: Procedure using PeQES for securely conducting empirical studies [81].

between  $B$ , the computation parties and the participants.  $R$ ,  $B$  and  $P_1, \dots, P_m$  are identical to their counterparts described in Section 5.1,  $S$  has the difference that its script for statistical analysis must now support execution as an SMPC through a protocol based on Shamir's secret sharing (e. g., BGW [17]). The script is expected not to leak sensitive results, i. e., not to perform reconstruction of values that contain personally identifiable information and of the result.  $B$ 's approval of  $S$  is assumed to ensure this. Further,  $R$  now possesses a key pair  $(R_{pk}, R_{sk})$ . We assume that all public keys are known to each entity. In practice, this can be accomplished by distribution through  $B$ .

- (1) In order to start the process,  $R$  sends  $S$  to  $B$ .
- (2) If  $B$  approves of  $S$ , it sends  $S$  to each computation party and  $L$ . Additionally, it provides them with  $\text{Sig}_{B_{sk}}(S, C_{1,pk}, \dots, C_{n,pk})$ , a signature over  $S$  and the computation parties' public keys.
- (3) Participants download  $S$  and  $\text{Sig}_{B_{sk}}(S, C_{1,pk}, \dots, C_{n,pk})$  from  $L$ . They can verify the signature in order to convince themselves that  $B$  has approved of the study and the computation parties.
- (4) After taking the survey, participant  $P_i$  splits their response  $S_i$  into secret shares  $S_{i,j}$  using a suitable secret sharing scheme (e. g.,  $(n, k + 1)$  Shamir's secret sharing if BGW is used). Each share  $S_{i,j}$  is encrypted using  $C_{j,pk}$ . The encrypted shares are sent to  $L$ .
- (5)  $L$  relays each encrypted secret share to the respective computation party, which can decrypt it using its secret key.
- (6) When the computation parties have received the response shares of all participants, they perform an SMPC in order to evaluate the script for statistical data analysis.

APPROACH

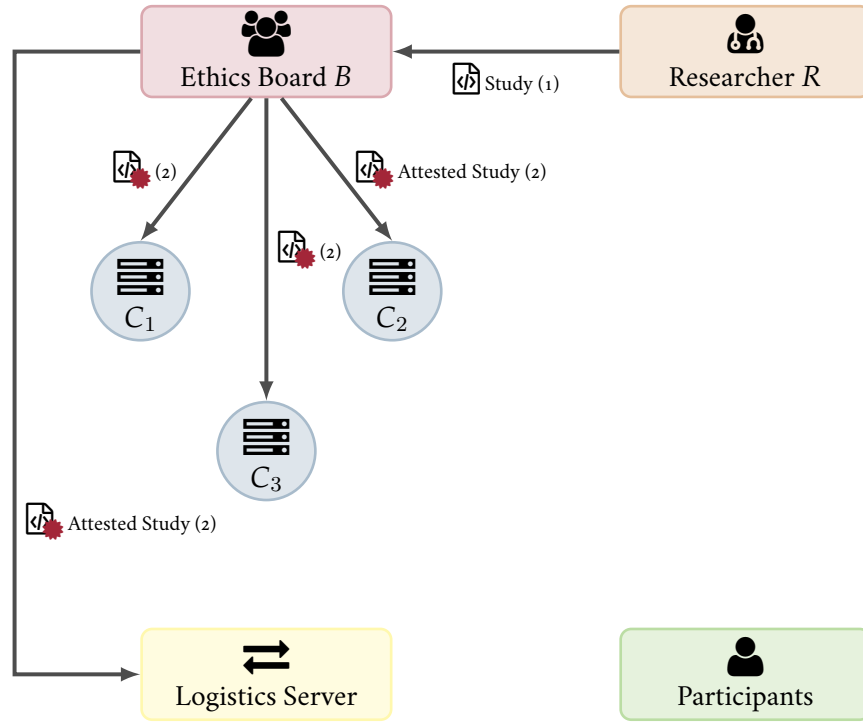


Figure 5.2: Publishing an empirical study on a platform based on SMPC (corresponding to steps (1) and (2)).

- (7) After the script for statistical data analysis has been executed, each computation party is left with one secret share of the result. These shares are encrypted using  $R_{pk}$  and sent to  $L$ .
- (8)  $L$  relays the encrypted result shares to  $R$ , who can decrypt them and reconstruct the result of the script for statistical data analysis.

This process (with  $n = 3$ ) is visualised in Figure 5.2 and Figure 5.3. Figure 5.2 shows the setup of the study, i. e., steps (1) and (2). Figure 5.3 shows the steps for participation in a study, execution of the script for statistical data analysis and transfer of the result shares to the researcher.

This protocol protects the participants' responses from illicit access if the SMPC is secure, i. e., a sufficient amount of computation parties are not corrupted. Further, as the computation parties are modelled as semi-honest, the security properties of correctness, independence of inputs, guaranteed output delivery and fairness are preserved. However, if a large enough subset of the platform providers  $\{H_1, \dots, H_n\}$  colludes and reads the secret-shared responses from memory, they can be reconstructed, compromising privacy. If BGW (see Subsection 2.2.4) is employed, a subset of  $\frac{n}{2}$  semi-honest computation parties suffices for this.

As noted above, we consider attacks mounted by participants to be out of scope for this work. For completeness, we mention a possible attempt to attack the protocol's independence of inputs. Consider the case of a coalition comprised of  $\geq \frac{n}{2}$  semi-honest computation parties and of  $q < m$  malicious participants (leaving  $m - q$  honest participants). After the  $m - q$  honest participants have submitted their responses, the semi-honest computation parties can reconstruct

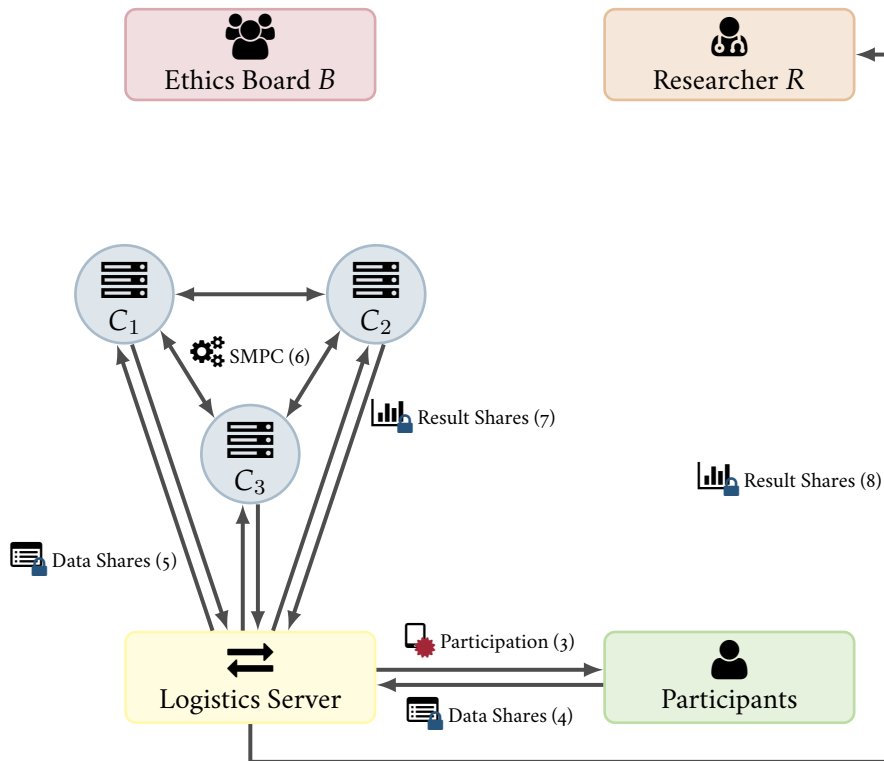


Figure 5.3: Steps (3) - (8) for securely conducting an empirical study through a platform based on SMPC.

them by combining the secret shares they have been provided with. The  $q$  malicious participants can now adjust their responses in order to influence the result of the script for statistical analysis (which they know through the semi-honest computation parties). This attack differs from a coalition of solely  $q$  malicious participants in that the attackers can now fine-tune the responses they submit as to evade detection by potential data cleaning steps in the script for statistical analysis.

We further note that the logistics server cannot compromise privacy, as it only sees encrypted data. However, it enhances the usability of the platform insofar as it allows participants to communicate with one instance only instead of the  $n$  computation parties.

### 5.3 MAIN CONTRIBUTION: COMBINING TEES AND SMPC

Section 5.2 introduces an approach for realising a platform providing confidentiality for empirical studies using SMPC. In this Section, we describe this work's main contribution. We combine PeQES [81] and its SMPC based sibling in order to achieve much stronger privacy guarantees than either system can provide on its own.

The entities relevant to this system are the same as in Section 5.2. Researcher  $R$  with key pair  $(R_{pk}, R_{sk})$  is again assumed to be fully malicious. They wish to conduct a study  $S$ , which consists of a survey and a script for statistical analysis through an SMPC. The ethics board  $B$  (with key pair  $(B_{sk}, B_{pk})$ ) and the study

participants  $P_1, \dots, P_m$  are modelled as described in Section 5.2. Further, there is a set of computation parties  $C_1, \dots, C_n$  with key pairs  $(C_{1,sk}, C_{1,pk}), \dots, (C_{n,sk}, C_{n,pk})$ , which are hosted by platform providers  $H_1, \dots, H_n$ . The computation parties can communicate securely with each other. Further, a logistics server  $L$  facilitates communication between the platform, the participants and the researcher.

The computation parties are, as described above, responsible for collecting the participants' answers, running a script for statistical analysis on these answers and returning its result to the researcher via  $L$ . However, they now protect the participants' answers and the execution of the script for statistical analysis by means of a suitable TEE (e. g., Intel SGX), i. e., they are implemented in such a way that shares of participants' data are never exposed to the platform providers. Additionally, the TEE's attestation capabilities are used to ensure correct execution of the script for statistical analysis.

Studies are conducted exactly as described in Section 5.2, however, step (2) is modified. Now,  $B$  performs a remote attestation with each of the computation parties  $C_i$  in order to convince itself that it is in fact running on secure hardware and has not been tampered with. Further,  $C_i$  is required to prove knowledge of  $C_{i,sk}$ , e. g., by signing a nonce provided by  $B$ . It is only after each computation party has successfully attested itself that  $B$  proceeds with signing the study and the computation parties' public keys and uploading it to the parties. By verifying  $\text{Sig}_{B_{sk}}(S, C_{1,pk}, \dots, C_{n,pk})$ , the computation parties convince themselves that the study has been approved by  $B$  and that all computation parties are running on trusted hardware.

When participating in a study, the participants are still required to verify  $\text{Sig}_{B_{sk}}(S, C_{1,pk}, \dots, C_{n,pk})$ . However, verification now assures them that

- $B$  has approved the study and checked the fact that the computation parties are protected by trusted hardware
- $C_j$  is in possession of  $C_{j,sk}$ , which ensures that no man-in-the-middle attack (e. g., by  $H_j$ ) is taking place.

In the following Section, we discuss the privacy guarantees provided by this platform and compare them to the privacy guarantees of PeQES and the platform described in Section 5.2.

### 5.3.1 CONSEQUENCES FOR PRIVACY GUARANTEES

In this Section, we compare the guarantees for confidentiality of survey responses of PeQES, the platform introduced in Section 5.2 and the combination of both described in Section 5.3. We consider correctness, independence of inputs, guaranteed output delivery and fairness as defined in Subsection 2.2.2 as out of scope and only analyse the privacy guarantees of the platforms. Without loss of generality we assume that BGW [17] using Shamir's secret sharing as proposed by Shamir [104] is used for SMPC. This protocol is secure against a semi-honest minority. Further, we assume that Intel SGX is employed as the utilised TEE.

The platform introduced in Section 5.2 only utilises SMPC for protecting the confidentiality of participants' responses. Every computation party is in possession of exactly one secret share of each participant's response. As BGW

utilises a  $(n, k + 1)$  Shamir’s secret sharing scheme with  $k = \lfloor \frac{n-1}{2} \rfloor$ , a set of  $k + 1$  distinct secret shares is required for reconstructing a response. As discussed in Subsection 2.2.4, BGW provides security (i. e., privacy, correctness, independence of inputs, guaranteed output delivery and fairness) in the presence of  $< \frac{n}{2}$  semi-honest attackers. Individual platform providers  $H_i$  hosting  $C_i$  cannot reconstruct responses on their own, even if they gain access to the data stored in  $C_i$ ’s memory, as it contains only one share of each response. The same is true for a coalition comprised of a semi-honest minority of platform providers, because they cannot acquire  $\frac{n}{2}$  secret shares of individual responses which are needed to reconstruct them. A coalition of  $\geq \frac{n}{2}$  semi-honest platform providers (which can read the computation parties’ memory and communication, but do not alter their behaviour or perform man-in-the-middle attacks), however, can reassemble the answers provided by participants, thus compromising privacy.

We further note that compute parties can easily be made malicious by the respective platform provider, e. g., through modification of the application code. However, this only affects correctness of the computation the script for statistical data analysis performs and guaranteed output delivery. Honest participants cannot be tricked into answering surveys not approved by the ethics board, as they can verify its signature via its public key (which we assume to be known in advance).

PeQES relies on the guarantees of Intel SGX for security. These guarantees encompass integrity and confidentiality for code and data protected by an enclave even in the presence of a privileged attacker. However, confidentiality only holds if the application processing the confidential data is not vulnerable to side-channel attacks, which SGX explicitly does not protect against. In PeQES, the script for statistical data analysis is provided in a DSL based on JavaScript and interpreted by the QuickJS<sup>2</sup> JavaScript engine. Even though interpreted scripts are less susceptible to side-channels than software executed as native code [118], the risk of such attacks can ultimately only be lowered by employing techniques to specifically counter this threat (e. g., by applying best-practices for secure coding). Further, SGX security guarantees may be circumvented if the script for statistical data analysis itself contains maliciously crafted code. The ethics board is trusted to reject studies containing such scripts. In order to make this verification practical, Meißner et al. [81] propose to limit the DSL in such a way that malicious intent is harder to conceal. For example, it is conceivable that the DSL only permits calling a set of whitelisted statistical functions, easing code review and making it difficult to hide malicious functionality. This, in turn, can also serve as a protection against side-channel attacks, because best-practices for side-channel resistant coding can be checked relatively effectively in an audit of a limited amount of code that is only allowed to call pre-audited functions. If, however, SGX is broken in the sense that the security guarantees Intel claims it has no longer hold, PeQES is no longer secure as well, even if it does not contain vulnerabilities itself. There have been attacks that demonstrated read-access to data stored in secure memory without the use of side-channels or software vulnerabilities in enclaves [26, 115]. While Intel has provided updates to the microcode of SGX-enabled processors to mitigate these specific attacks, the emergence of novel, similarly critical vulnerabilities in the future cannot be ruled out.

<sup>2</sup> <https://bellard.org/quickjs/>

## APPROACH

Our approach uses both SMPC (more specifically, BGW) and Intel SGX to guarantee confidentiality of survey responses. Compromising its security requires controlling a potent coalition of platform providers, which are all able and willing to read data from protected memory either by using side-channel attacks or by circumventing the TEE's security mechanisms.

An overview of different attack scenarios and privacy preservation of the different platforms under these scenarios is given in Table 5.1. As described above, a platform using only Intel SGX (e. g., PeQES) to protect privacy fails if either the enclaved application is vulnerable to side-channel attacks or if SGX itself is attackable. A platform using SMPC (BGW) to protect privacy is vulnerable in the presence of  $\geq \frac{n}{2}$  colluding attackers. Our proposed platform using both of these techniques protects privacy in every scenario with either an honest majority of computation parties or in which attackers cannot read protected memory, effectively providing an additional line of defence compared to both PeQES and the platform proposed in Section 5.2. In the event that Intel SGX turns out to be susceptible to novel attacks or that side-channels are found in either the platform application or the script for statistical analysis, the fact that user data are stored in a decentralised way protects them from unauthorised access. In this case, an adversary would need to be in control of enough computation parties. As the parties are hosted by independent platform providers, this is unlikely. In reality, platform providers can be different research institutions. Such institutions are concerned for their reputation and are often in a competitive relationship, incentivising them to monitor each other for scientific misconduct and lowering chances of malicious collusion.

In the reverse case, i. e., in the event that Intel SGX in combination with a securely coded platform application protects survey responses reliably, even a coalition of all computation parties (e. g., through collusion of the respective institutions or infection by malware) cannot access private data.

The additional layer of security present in the proposed platform strengthens privacy guarantees, however, it also introduces performance overhead. In order to investigate its impact, we implement a prototype of the platform, measure the duration of various computations and compare them to platforms with less strong privacy guarantees. The implementation is described in Chapter 6; the evaluation and its results are found in Chapter 7.

SGX vulnerable	Side-chan. exploitable	# Attackers	Only SGX	Only SMPC	SMPC + TEE
No	No	$< n/2$	✓	✓	✓
		$\geq n/2$		✗	✓
	Yes	$< n/2$	✗	✓	✓
		$\geq n/2$		✗	✗
Yes	No	$< n/2$	✗	✓	✓
		$\geq n/2$		✗	✗
	Yes	$< n/2$		✓	✓
		$\geq n/2$		✗	✗

*Table 5.1:* Different attack scenarios a platform for conducting empirical studies can face. The left-hand side specifies the capabilities and, if applicable, the number of colluding computation parties. The right-hand side lists whether a platform making use of either only SGX, only SMPC or both to protect private data can guarantee privacy (✓) or not (✗).





## IMPLEMENTATION

---

In order to assess the performance of the system proposed in Section 5.3, we implemented a prototype application that realises a combination of SMPC and TEEs as above. Further, we implemented prototypes realising platforms secured only by SGX (resembling PeQES [81]), only by SMPC (resembling the platform proposed in Section 5.2) and neither technology as a baseline. Code artefacts have been made available online<sup>1</sup>. In this Section, we introduce the software libraries and frameworks the prototypes are built upon. In Chapter 7, we describe the experiments used to evaluate the performance of the prototypes, the results of which we present in Chapter 6.

### 6.1 REQUIREMENTS

The prototype of the platform we propose is a proof-of-concept meant to compare computational efficiency of the proposed systems. It is specifically *not* designed to provide the full functionality of the system proposed in Section 5.3, but only serves to assess the feasibility of our approach in terms of performance. We identify several requirements that the prototype is expected to fulfil.

First, the prototype is expected to implement a distributed system that is able to perform an SMPC. Shares of the inputs, intermediary results and results must be protected by trusted hardware. We chose Intel SGX as the TEE the platform uses because it is widely available on current hardware. Second, in order to enable participants to split their responses into secret shares, the protocol used for SMPC must be based on secret sharing. Further, it must support computations between more than two parties and be secure against a semi-honest minority of attackers. We explicitly exclude functionality that does not affect computational performance, i. e., the ethics board, the researcher and verification of signatures.

In order to compare the performance of our approach to platforms using only SGX, only SMPC or neither of those to protect user data, we also implement prototypes realising them.

### 6.2 CHOICE OF TECHNOLOGY

For implementing the prototypes described above, we make use of existing software frameworks and libraries. In order to be able to perform SMPCs, we utilise a suitable library implementing such primitives. To make our prototypes compatible with Intel SGX, we are faced with the choice of either implementing a native SGX application or using technologies that enable running unmodified applications in SGX. In this Section, we describe these choices.

---

<sup>1</sup> <https://gitlab-vs.informatik.uni-ulm.de/theses/2022-ma-dispan/-/tree/v1.1/code>

---

```

1 inputs = mpc.input(in_value)
2 product = secint(1)
3
4 for x in inputs:
5     product = product * x
6 # alternatively:
7 # product = mpc.prod(inputs)
8
9 result = await mpc.output(product)
10 print(result)

```

---

*Listing 6.1:* Computing the product of the input values in MPyC [101].

### 6.2.1 TECHNOLOGY FOR SMPC

In Section 3.1, we introduce three different software libraries that implement SMPC, namely Jiff, MP-SPDZ and MPyC. After careful comparison of these three libraries we conclude that Jiff is suited best for implementing the proof-of-concept application. In the following, we discuss the reasons for this decision.

MPyC is focused on being user-friendly and working *out of the box* without requiring much configuration. Notably, it makes use of operator overloading, which leads to SMPC code that is nearly identical to regular Python code (as shown in Listing 6.1<sup>2</sup>). The system we propose requires researchers that conduct empirical studies to write scripts for statistical analysis. Using a library like MPyC, which would allow writing SMPC scripts without requiring expertise in SMPC may boost acceptance of our platform. However, MPyC’s user-friendliness comes at the price of limited flexibility. For example, it does not have built-in support for parties that only provide inputs but do not stay online during the computation itself (which is required for modelling study participants). While achieving such a feature is technically possible with MPyC, it requires actively programming *against* the library’s abstractions, discouraging their usage. For this reason, we do not use MPyC for implementing the prototype.

MP-SPDZ is the opposite of MPyC in the sense that it provides flexibility at the cost of usability. It supports a large number of protocols for SMPC, some of which are based on Shamir’s secret sharing. Further, it permits asynchronous input parties, which leave the computation right after sharing their input. However, it has a rather steep learning curve. As shown in Listing 6.2, it uses a DSL that introduces novel syntax to Python (e. g., for looping under SMPC) and requires using specialised functions for common operations (e. g., `print_ln` instead of `print`).

Jiff has native support for asynchronous input parties and implements an SMPC protocol secure against a semi-honest minority of attackers. It is written in JavaScript and SMPCs are expressed as regular method calls as shown in Listing 6.3<sup>3</sup>, which contains code using Jiff to compute the product of the parties’ inputs. It thus does not require learning new syntax for expressing computations

<sup>2</sup> The example code is inspired by Hastings et al. [50] (See: <https://github.com/MPC-SoK/frameworks/blob/master/mpyc/source/mult3.py>) and Schoenmakers [100] (See: <https://mpyc.readthedocs.io/en/latest/demos.html#oneliners-py>).

<sup>3</sup> The example shown is a modified version of code by Hastings et al. [50] (See: <https://github.com/MPC-SoK/frameworks/blob/master/jiff/source/mult3/mpc.js>).

---

```

1 prod = Array(1, sint)
2 prod[0] = sint(1)
3
4 @for_range(number_clients)
5 def loop_body(i):
6     a = sint.get_input_from(i)
7     prod[0] = prod[0] * a
8
9 print_ln('%s', prod[0].reveal());

```

---

*Listing 6.2:* Computing the product of the input values in MP-SPDZ [61].

---

```

1 var shares = jiff_instance.share(input);
2 var product = shares[0];
3 for (var i = 1; i < jiff_instance.party_count; i++) {
4     product = product.smult(shares[i]);
5 }
6 console.log(jiff_instance.open(product));

```

---

*Listing 6.3:* Computing the product of the input values in Jiff [84].

while retaining flexibility. Further, it has a unique architecture that makes use of a central logistics server, which provides resilience against network failures and permits compute parties to participate in the computation without needing a public IP address. However, the potentially malicious logistics server enables novel types of attackers which need to be considered should Jiff be chosen for deployment in a productive setting. Further, it is not clear whether or not Jiff is currently maintained. In its current state, it is fit to conduct SMPC, however, bug reports and pull requests are not answered by the developers.

Nevertheless, we deem Jiff the most suitable library for providing SMPC capabilities to the prototype. The reason for this choice is that in our opinion, Jiff is the easiest-to-use library for SMPC that we encountered. In particular, asynchronous input parties are very straight-forward to implement in Jiff, whereas MP-SPDZ and MPyC require considerable efforts to achieve the same. Additionally, Jiff is written in JavaScript and thus able to run in modern web browsers. In particular, a web application used by participants for submitting survey responses could directly utilise Jiff for sharing confidential data. MP-SPDZ, MPyC and all other technologies for SMPC the authors of this work are aware of do not support execution in a current browser and would require installation of additional software.

We discuss the version and configuration of Jiff that we employ in Subsection 6.2.3.

### 6.2.2 TECHNOLOGY FOR SGX

In order to make use of Intel SGX, one has several options. First, it is possible to develop native SGX applications that are hand-written to use SGX features. This approach gives developers maximum control and allows a fine-grained division of the application into enclave code and non-enclave code. However, it also requires expert knowledge, and the resulting SGX-enabled applications are

restricted to SGX-enabled platforms. In order to write native SGX applications, one typically makes use of a *Software Development Kit* (SDK), which makes SGX' features available in a high-level programming language. Notable examples for SGX SDKs are the Intel SGX SDK<sup>4</sup> (C/C++), the Open Enclave SDK<sup>5</sup> (C/C++), Fortanix EDP<sup>6</sup> (Rust), Teaclave<sup>7</sup> (Rust, Python, Java [82]) and EGo<sup>8</sup> (Go).

Further, there are solutions aiming to enable running unmodified applications in SGX. Some of these solutions (e. g., Gramine-SGX, which was discussed in Subsection 3.2.1, but also Mystikos<sup>9</sup>, Occlum<sup>10</sup> or SGX-LKL<sup>11</sup>) are or resemble library OSes, others provide runtimes that securely run interpreted programming languages (e. g., ScriptShield, see Subsection 3.2.2, for JavaScript, Lua and Squirrel, or Enarx<sup>12</sup> for WebAssembly). This approach reduces requirements in terms of development effort and expertise in Intel SGX, however, it comes at the cost of a larger TCB and less computational performance.

In order to utilise Jiff, we must develop our application in JavaScript, and thus utilise a technology that allows us to run a JavaScript interpreter inside SGX enclaves. In Section 3.2, we investigated Gramine-SGX and ScriptShield, both of which meet this requirement. ScriptShield, however, states<sup>13</sup> that it requires *Ubuntu 16.04*, which is outdated. Additionally, it does not seem to be actively maintained and we were unable to compile it on our test system. In comparison, Gramine-SGX is under active development and has reached a high level of maturity. It has excellent documentation and is used in a number of productive systems<sup>14</sup>. For these reasons, we choose Gramine-SGX for implementing the SGX-based prototypes.

### 6.2.3 VERSIONS

We use Jiff in the latest available version, i. e., commit 8ea565d<sup>15</sup>. As previously mentioned, parties in Jiff communicate via encrypted channels.

We use node.js in version v18.14.1 for interpreting JavaScript code. In order to run the prototypes on protected hardware, we use Gramine-SGX in version 1.4 with a manually applied patch<sup>16</sup>

---

4 <https://github.com/intel/linux-sgx>  
5 <https://openenclave.io/sdk/>  
6 <https://edp.fortanix.com/>  
7 <https://teaclave.apache.org/>  
8 <https://www.edgeless.systems/products/ego/>  
9 <https://github.com/deislabs/mystikos>  
10 <https://occlum.io/>  
11 <https://github.com/llds/sgx-lkl>  
12 <https://enarx.dev/>  
13 See: <https://github.com/OSUSecLab/scriptshield>  
14 See: <https://gramine.readthedocs.io/en/latest/gramine-users.html>  
15 <https://github.com/multiparty/jiff/tree/8ea565d3d0becde8f71243fb9daea6ef0ba9bb7e>  
16 The patch we manually applied can be found under <https://github.com/gramineproject/gramine/pull/1203>. It does not change functionalities in Gramine, but fixes a bug that hinders installation. After we performed our experiments, the patch has been merged into Gramine's main branch, presumably making it part of the upcoming release.

## EVALUATION

---

In this Section, we describe the experimental apparatus used to evaluate the performance of the platform for securely conducting empirical studies described in Section 5.3. Section 7.1 describes the methodology of our experiments. Section 7.2 gives the results of the evaluation.

### 7.1 METHODOLOGY

We implemented four different platform prototypes:

- **Naive:** A platform that is able to receive survey responses from study participants and run a script for statistical analysis on them, but neither uses a TEE nor SMPC to protect their privacy.
- **SGX-only:** A platform similar to PeQES (see Section 5.1) in that it uses a TEE to ensure privacy of survey responses.
- **SMPC-only:** A platform as described in Section 5.2, in which survey responses are secret-shared between several computation parties and which uses SMPC to perform computations on them.
- **SMPC-SGX:** A platform that uses both SMPC and a TEE to protect confidentiality of survey responses during storage and computation (see Section 5.3).

The architectures of the implemented prototypes are shown in Figure 7.1. *Naive* is implemented as a JavaScript application running on `node.js`. In the prototype *SGX-only*, this application runs in an SGX enclave using Gramine-SGX. The participants are implemented as `curl`<sup>1</sup> commands, which submit their responses directly to the platform using HTTP POST requests. In *SMPC-only* and *SMPC-SGX*, the platform is realised as  $n$  compute parties (in Figure 7.1 with  $n = 3$ ). These parties, the logistics server and the participants are JavaScript applications that use Jiff for secret sharing and SMPC. In *SMPC-SGX*, the computation parties run in SGX enclaves using Gramine-SGX. We note that in a real-world setting, participants would enter their survey responses via a web application delivered through a web browser. Our approach of simulating participants through command-line applications does not lead to distorted performance measurements, as it uses the same REST-API for communication as potential web applications would.

In the following, we describe the configurations of Jiff and Gramine that we use for our experiments. In all experiments involving *SMPC-only* or *SMPC-SGX*, we set  $n = 3$ . This is a sensible choice, as this number of computation parties is low enough to be used in practice, but large enough to tolerate one corrupted party. We use Jiff's *BigNumber*, *FixedPoint* and *Negative Number* extensions, which enable computations using negative and fixed-point numbers. We instruct Jiff to use an accuracy of 5 integer digits and 2 decimal digits, requiring us to set  $p$  to a

---

<sup>1</sup> <https://curl.se/>

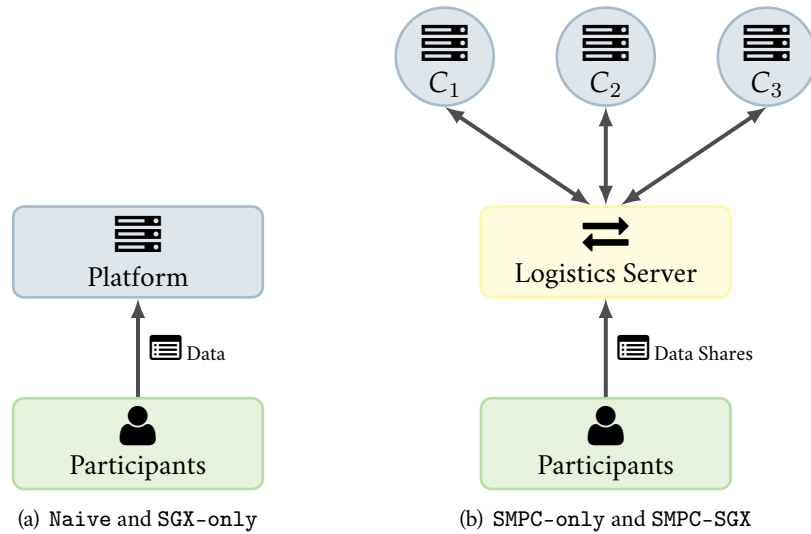


Figure 7.1: Architectures of the implemented prototypes. (a) shows the architecture of Naive and SGX-only. Here, participants send their responses directly to the platform. (b) shows the architecture of SMPC-only and SMPC-SGX. Here, participants send secret shares of their response to the logistics server, which relays them to the respective computation party. The computation parties communicate with each other via the logistics server.

prime with at least 14 digits. We thus choose  $p = 260\,623\,316\,657\,987$ , setting the finite field the SMPCs are performed in to  $\mathbb{Z}_{260\,623\,316\,657\,987}$ . All entities communicate with the logistics server using a REST API provided by Jiff through Jiff’s *RestAPIServer* extension. Unless stated otherwise, the logistics server acts as a *crypto provider*, i. e., it supplies the computation parties with Beaver triplets used to facilitate more efficient multiplication (see Subsection 2.2.5).

Further, we note that we disable Jiff’s transport encryption for the experiments. The reason for this is that Jiff uses *libsodium.js*<sup>2</sup> for performing asymmetric cryptography. Unfortunately, *libsodium.js* in version 0.7.9 seems to be incompatible with Gramine-SGX, as attempts to use both technologies in conjunction result in errors which we could not resolve. For fairness, we also use no encryption with Naive and SGX-only and note that the choice of not using encryption does not change the relative results of the performance evaluation.

The two enclaved prototypes SGX-only and SMPC-SGX make use of Intel SGX through Gramine-SGX. In both cases, the respective enclave is built to only contain *node.js* and its dependencies. The code implementing the platform is loaded at runtime, where its integrity is verified by Gramine-SGX. We specify the enclaves’ EPC size as 4 GB, which is necessary because *node.js* requires a relatively large amount of memory.

The experiments are performed on one server machine running *Ubuntu 22.04.2 LTS*. It possesses an *Intel Xeon E-2186G* CPU with 6 cores and 64 GB of RAM. Each experiment that we perform is conducted in the following manner: First, the logistics server is launched (omitted in the case of Naive and SGX-only). Afterwards, the platform/computation parties are started. When startup has completed, the participants start submission of their inputs concurrently. After all  $m$  participants have submitted their input, the platform/com-

<sup>2</sup> <https://www.npmjs.com/package/libsodium-wrappers>

putation parties execute the script for statistical analysis. Diverging from the protocol proposed in Section 5.2 and Section 5.3, computation parties in the SMPC-only and SMPC-SGX prototypes perform a reconstruction of the result using the result shares in order to measure the performance of this process.

During the experiments, various measurements are performed. The runtime of the script for statistical analysis is called the *compute time*. For SMPC-only and SMPC-SGX, we measure the time taken by the participants to split the response into secret shares and the time taken by the computation parties to reassemble the result, which we call *share time* and *open time*. In some experiments, the computation parties engage in preprocessing, generating Beaver triplets. The time taken by preprocessing is called *preprocessing time*.

We exclude measuring the duration of starting enclaves, because this process is only performed once per study per platform/computation party and does typically not take longer than a few seconds. The reason for this startup delay is the fact that old implementations of Intel SGX, including the one in our test machine, do not support dynamic memory allocation. Instead, the maximum amount of memory the enclave is permitted to own must be allocated at startup time, which is 4 GB in our case. Recent incarnations of SGX-enabled processors support *Enclave Dynamic Memory Management* (EDMM) [80], which alleviates this issue and allows for faster startup times.

If not specified otherwise, the measured values presented in Section 7.2 are calculated as the mean of running the respective experiment 100 times. Note that some experiments yield multiple measurements, e. g., running an experiment with the prototype SMPC-only yields three computation time measurements, because each of the three computation parties measures this value separately.

**Script for Statistical Analysis** Each participant submits one natural number to the computation. Unless stated otherwise, the script for statistical analysis used in our performance measurements performs a two-sample t-test [36], which is a regularly used primitive in statistical analyses. A two-sample t-test is performed by evaluating

$$T_2^* = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{\frac{\text{Var}(X_1)}{n_1} + \frac{\text{Var}(X_2)}{n_2}}} \quad (7.1)$$

where  $X_1$  and  $X_2$  are vectors consisting of the inputs by parties with odd and even IDs respectively,  $n_1$  and  $n_2$  their respective sizes,  $\bar{X}$  the mean of  $X$  and  $\text{Var}(X)$  the variance of  $X$ . Given a sample  $X = (x_1, \dots, x_{n_X})$ , the variance of  $X$  is defined as

$$\text{Var}(X) = \frac{1}{n_X - 1} \sum_{i=1}^{n_X} (x_i - \bar{x})^2 \quad (7.2)$$

However, this formula is not ideal for SMPC because it requires  $n_X$  expensive multiplications to be evaluated. The alternative formulation

$$\text{Var}(X) = \frac{n_X \sum_{i=1}^{n_X} x_i^2 - (\sum_{i=1}^{n_X} x_i)^2}{n_X^2} \quad (7.3)$$

is better-suited, because it can be optimised more effectively [85]. Each value in  $X$  is the input of one survey participant. We note that the formula requires each input squared, however, squaring under SMPC is expensive. For this reason, we

instruct the participants to not only provide a secret shared  $x_i$  as input, but also its secret shared square  $x_i^2$ . Notably, this does not leak information about  $x_i$  and saves  $n_X$  multiplications. Further, we observe that Equation 7.1 involves evaluating a square root, which is an expensive operation under SMPC as well. For this reason, we perform the t-test by first evaluating

$$(T_2^*)^2 = \frac{(\bar{X}_1 - \bar{X}_2)^2}{\frac{\text{Var}(X_1)}{n_1} + \frac{\text{Var}(X_2)}{n_2}} \quad (7.4)$$

under SMPC and calculating  $\sqrt{(T_2^*)^2} = T_2^*$  locally<sup>3</sup>. As  $\sqrt{\cdot}$  and  $(\cdot)^2$  are reversible, this does not leak information about the inputs that would not have been evident from the original computation. In order to give a fair comparison, all prototypes perform the t-test as portrayed above.

We conduct the following experiments:

- E1:** Measure compute time and open time for a varying number of survey participants, i. e., for  $m \in \{5, 10, 50, 100, 500, 1000\}$ . Compute time is measured for all prototypes, open time is only measured in SMPC-only<sup>4</sup>
- E2:** Considering a setting in which the logistics server does not act as a crypto provider, measure compute time and preprocessing time for a varying number of survey participants, i. e., for  $m \in \{5, 10, 50, 100, 500, 1000\}$ .
- E3:** Measure the compute time in a setting with  $m = 100$  while varying the workload. The workload is varied by having the script for statistical analysis calculate  $u$  t-tests for  $u \in \{1, 2, 4, 8\}$ .
- E4:** Measure the share time in a scenario where each participant inputs  $\ell$  natural numbers into the computation for  $\ell \in \{5, 10, 50, 100, 500, 1000, 5000, 10\,000\}$ .

Experiments E1 to E3 are performed using the platform prototypes we implemented. Experiment E4 does not depend on the platform and is run in an offline setting.

## 7.2 RESULTS

In this Section, we present the results of our experiments.

### 7.2.1 VARYING THE NUMBER OF SURVEY PARTICIPANTS

In experiment E1, we compare the four implemented prototypes with regards to compute time. Further, we measure the time to reconstruct shared secrets (open time) in SMPC-only. In order to determine the impact of the number of survey

<sup>3</sup> In reality, calculation of the square root is performed by the researcher after reconstructing the result of the SMPC via the secret shares they received. In our prototype, the computation parties perform this step after reconstructing the result.

<sup>4</sup> We only measure open time in a non-SGX setting, because in a real-world setting, this step is performed by researcher  $R$ , presumably via a non-enclaved application.



participants  $m$  on the measured durations, we vary it. Specifically, we perform measurements for each  $m \in \{5, 10, 50, 100, 500, 1000\}$ . The measured compute times of this experiment can be found in Table 7.1 and Figure 7.2. The open times are visualised in Figure 7.3.

Considering the compute times, we observe that the prototypes not relying on SMPC outperform the others by several orders of magnitude. This is expected for several reasons: first, SMPC requires communication, which is expensive in terms of performance. Second, SMPC is realised using Jiff, which loads a large amount of dependencies into memory, potentially fostering page-faults. Third, we use Jiff with all extensions applied (BigNumber, FixedPoint and Negative), which add additional overhead compared to the non-SMPC-based prototypes that operate on unboxed numbers.

SGX-only is  $\approx 4 - 6\times$  slower in its computation phase as Naive. This is in line with the overhead imposed by Gramine-SGX as reported by its authors (see Subsection 3.2.1). In comparison, SMPC-SGX is  $\approx 10\times$  slower than SMPC-only. We explain the reasons for this behaviour in Subsection 8.1.1.

Further, we observe that compute time only increases slightly as  $m$  grows. This is because  $m$  only influences the number of additions performed under SMPC. The number of multiplications and divisions remains unchanged. Notably, addition incurs no communication cost, which means that increasing  $m$  does not increase the number of messages sent. It only increases local computation, which impacts computation time only slightly. Further, the time to open, i. e., reconstruct a secret-shared value from shares, is not influenced by  $m$ .

### 7.2.2 PREPROCESSING IN SMPCS

This Section gives the results of experiment E2. This experiment is performed only for SMPC-only and SMPC-SGX, because the other prototypes do not make use of SMPC. It differs from the other experiments in that the computation parties perform a preprocessing phase. In this scenario, the logistics server does not act as a crypto provider. The mean and standard error of preprocessing time and compute time using Jiff's preprocessing feature are shown in Table 7.2. We note that the results shown are the mean of 10 (as opposed to 100) experiment runs, the reason for which are the relatively long run times of this experiment.

For prototype SMPC-only, the use of preprocessing shifts part of the work performed by the computation parties from the computation phase to the preprocessing phase. We observe that the compute time has been reduced by  $\approx 50\%$ . The reason for this is that the computation parties no longer need to query the logistics server for Beaver triplets during the computation phase, which reduces communication overhead. However, preprocessing requires a considerable amount of time before computation can begin. In particular, preprocessing time is longer than the time saved during the computation phase. The reason for this is that preprocessing requires  $O(n^2)$  messages, whereas querying the server for Beaver triplets only requires  $O(n)$  messages. The increase in preprocessing time as  $m$  grows is unexpected at first sight, as the amount of preprocessing to be performed does not depend on  $m$ . It is, however, explained with the fact that in this experiment, the participants begin sending their inputs *before* preprocessing has finished, placing additional load on the computation parties.

EVALUATION

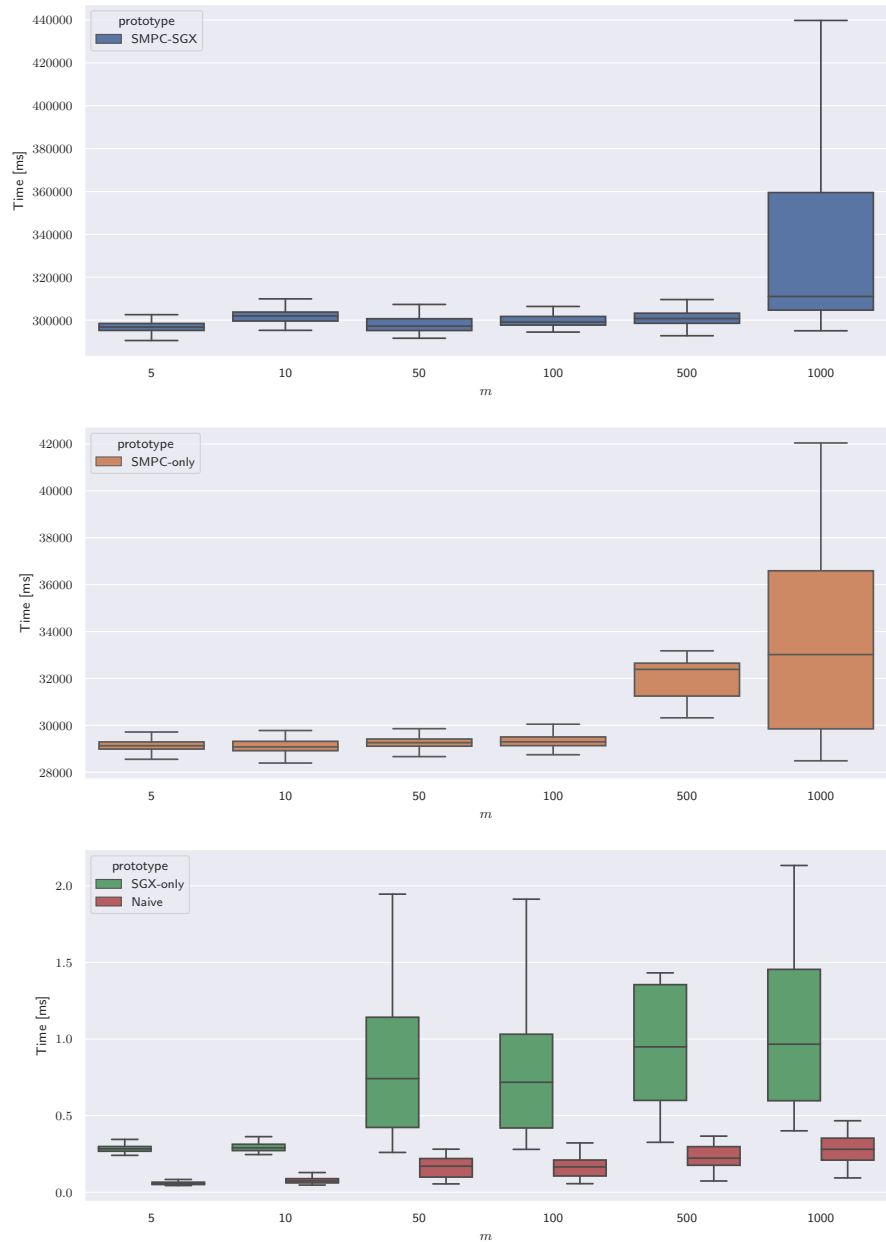


Figure 7.2: Results of experiment E1 with regards to compute time. Note the three box plots reside in different sections of the y-axis.

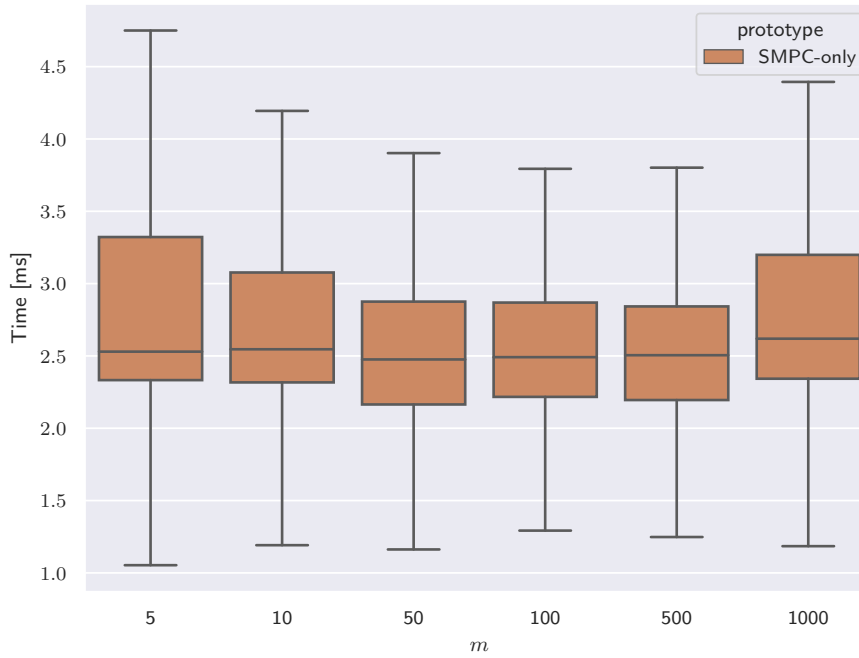
$m$	Naive	SGX-only
5	$0.06 \pm 0.00$ ms	$0.29 \pm 0.00$ ms
10	$0.08 \pm 0.00$ ms	$0.31 \pm 0.01$ ms
50	$0.17 \pm 0.01$ ms	$0.80 \pm 0.04$ ms
100	$0.16 \pm 0.01$ ms	$0.76 \pm 0.04$ ms
500	$0.22 \pm 0.01$ ms	$0.94 \pm 0.04$ ms
1000	$0.27 \pm 0.01$ ms	$1.03 \pm 0.05$ ms

(a) Naive and SGX-only

$m$	SMPC-only	SMPC-SGX
5	$29.13 \pm 0.01$ s	$296.95 \pm 0.17$ s
10	$29.11 \pm 0.02$ s	$302.00 \pm 0.22$ s
50	$29.27 \pm 0.01$ s	$297.84 \pm 0.21$ s
100	$29.32 \pm 0.02$ s	$299.28 \pm 0.28$ s
500	$32.00 \pm 0.04$ s	$301.56 \pm 0.31$ s
1000	$33.33 \pm 0.21$ s	$333.73 \pm 2.51$ s

(b) SMPC-only and SMPC-SGX

*Table 7.1:* Compute times of the different prototypes as measured in experiment E1. We report the mean value and standard error per experimental configuration. The results for Naive and SGX-only are shown in (a), the results for SMPC-only and SMPC-SGX are shown in (b).



*Figure 7.3:* Open times in SMPC-only, measured over the course of experiment E1. The open times are relatively constant as  $m$  increases.

## EVALUATION

$m$	preprocessing	compute
5	$64.97 \pm 0.21$ s	$16.70 \pm 0.12$ s
10	$64.51 \pm 0.14$ s	$16.85 \pm 0.10$ s
50	$65.91 \pm 0.13$ s	$16.83 \pm 0.08$ s
100	$67.19 \pm 0.09$ s	$16.68 \pm 0.07$ s
500	$80.52 \pm 0.22$ s	$16.73 \pm 0.10$ s
1000	$100.96 \pm 0.17$ s	$16.66 \pm 0.06$ s

(a) SMPC-only

$m$	preprocessing	compute
5	$2215.51 \pm 50.27$ s	$1175.90 \pm 52.17$ s
10	$2298.04 \pm 33.98$ s	$1260.83 \pm 36.51$ s
50	$2316.50 \pm 50.22$ s	$1250.29 \pm 31.72$ s
100	$2353.87 \pm 39.19$ s	$1219.31 \pm 48.46$ s
500	$2542.64 \pm 24.37$ s	$1290.32 \pm 22.76$ s
1000	$2498.52 \pm 39.24$ s	$1362.04 \pm 17.82$ s

(b) SMPC-SGX

Table 7.2: Preprocessing and compute times measured throughout experiment E2. (a) shows the results for prototype SMPC-only, (b) shows the results for SMPC-SGX. We provide mean and standard error values.

SMPC-SGX has a different behaviour when preprocessing is used. Preprocessing time is  $\approx 2000$  s, compute time is  $\approx 1300$  s. In Experiment E1, without preprocessing, SMPC-SGX is  $\approx 10\times$  slower than SMPC-only. In this experiment, this factor is increased considerably. Preprocessing in SMPC-SGX is  $\approx 30\times$  slower than in SMPC-only; for computation this factor is  $\approx 80$ . We suspect that the reason for this behaviour lies in the increased memory usage caused by preprocessing. Instead of acquiring correlated random numbers (e. g., Beaver triplets) on-demand from the logistics server, the computation parties generate them in advance and store them until needed. The storage required for these numbers must not be underestimated. As we use Jiff’s BigInteger extension, each value has a non-trivial memory footprint on its own. Additionally, the computation we perform requires a large amount of correlated random numbers, e. g., one single secure division requires 8780 Beaver triplets and 17 566 other correlated, secret-shared numbers. In light of the fact that EPC size is very limited in pre IceLake architectures such as our test system’s, we assume the cost of page swapping in SGX to be the culprit for these differences in performance.

## 7.2.3 VARYING THE WORKLOAD

In experiment E3, we measure computation time as we vary the workload executed by the prototype platforms. For this, we define  $u$  as the number of t-tests calculated by the script for statistical analysis and perform measurements of the

$m$	Naive	SGX-only
1	$0.26 \pm 0.01$ ms	$1.19 \pm 0.05$ ms
2	$0.26 \pm 0.01$ ms	$1.18 \pm 0.05$ ms
4	$0.29 \pm 0.01$ ms	$1.27 \pm 0.06$ ms
8	$0.39 \pm 0.01$ ms	$1.58 \pm 0.06$ ms

(a) Naive and SGX-only

$m$	SMPC-only	SMPC-SGX
1	$29.45 \pm 0.08$ s	$291.94 \pm 0.43$ s
2	$46.31 \pm 0.03$ s	$711.87 \pm 0.39$ s
4	$88.57 \pm 0.05$ s	$2361.82 \pm 2.31$ s
8	$175.11 \pm 0.09$ s	$6487.31 \pm 7.24$ s

(b) SMPC-only and SMPC-SGX

*Table 7.3:* Compute time in the four prototypes when workload is increased. (a) shows the mean and standard error of Naive and SGX-only’s compute times; (b) shows the same metrics for SMPC-only and SMPC-SGX.

compute time for all  $u \in \{1, 2, 4, 8\}$ . Note that in this experiment, the survey participants do not alter the amount of inputs they send to the platform. Rather, the platform performs multiple iterations calculating t-tests reusing the participants’ inputs repeatedly. The results of these experiments are given in Table 7.3 and Figure 7.4. Compute times using Naive and SGX-only are negligible even for  $u = 8$ . SMPC-only’s compute time clearly shows the linear growth we expect it to have as  $u$  is increased. SMPC-SGX’s compute times grow faster. This behaviour is somewhat unexpected, however, we conjecture that it can be explained through the facts that Jiff sends additional management messages (*heartbeats*) in longer-running applications and that an increased workload increases memory usage, resulting in more time spent on page swapping.

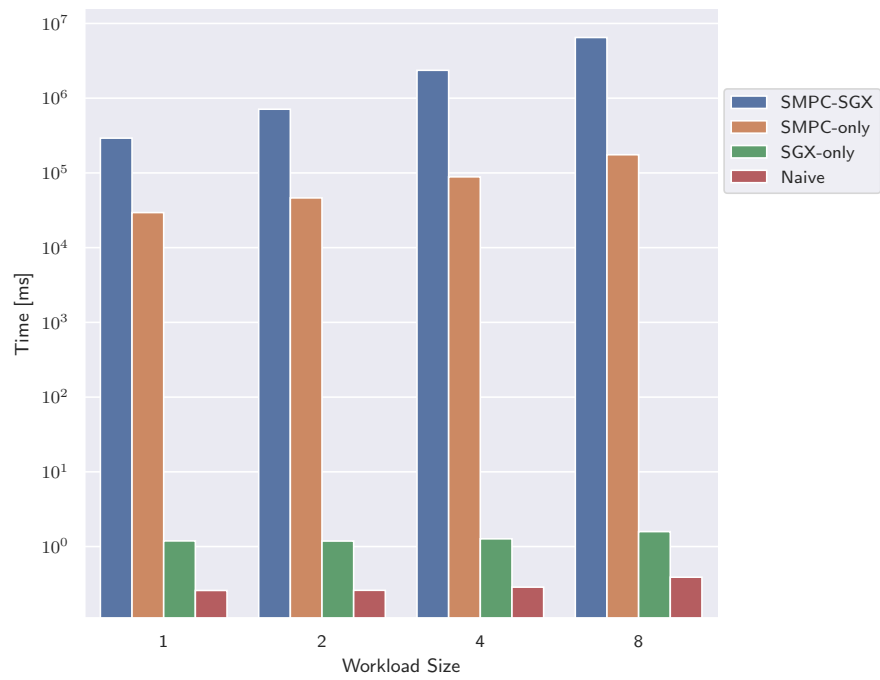
#### 7.2.4 VARYING THE INPUT LENGTH

In experiment E4, we investigate the performance impact of the survey’s size for participants. This experiment is not conducted using the platform prototypes, but using a script that utilises Jiff’s API for splitting values into secret shares. More specifically, the script splits an array of size  $\ell$  into secret shares using a (3, 2) Shamir’s secret sharing scheme using Jiff’s `share_array`<sup>5</sup> function. In order to assess the impact of extensions for supporting arbitrarily large numbers, fixed-point numbers and negative numbers, we perform the experiment multiple times progressively applying more extensions.

The results are given in Table 7.4. Generally speaking, applying extensions lowers performance. With all built-in extensions applied, sharing is about an

<sup>5</sup> [https://multiparty.org/jiff/docs/jsdoc/module-jiff-client-JIFFClient.html#share\\_array](https://multiparty.org/jiff/docs/jsdoc/module-jiff-client-JIFFClient.html#share_array)

## EVALUATION



*Figure 7.4:* Compute time in the four prototypes when workload is increased. Note that the y-axis is scaled logarithmically.

order of magnitude slower than without any extensions. Further, the time required for secret sharing scales linearly with  $\ell$  and is below one second even for  $\ell = 10\,000$  with all built-in extensions applied.

$\ell$	No ext.	BigNumber
5	$1.50 \pm 0.01$ ms	$5.76 \pm 0.01$ ms
10	$1.89 \pm 0.01$ ms	$7.38 \pm 0.04$ ms
50	$2.82 \pm 0.02$ ms	$17.64 \pm 0.11$ ms
100	$3.90 \pm 0.03$ ms	$25.00 \pm 0.19$ ms
500	$12.65 \pm 0.07$ ms	$59.55 \pm 0.42$ ms
1000	$22.88 \pm 0.09$ ms	$95.75 \pm 0.48$ ms
5000	$78.59 \pm 0.48$ ms	$360.32 \pm 0.69$ ms
10 000	$142.17 \pm 0.45$ ms	$675.78 \pm 1.39$ ms

(a) No extensions (left) and BigNumber extension (right).

$\ell$	+FixedPoint	+Negative
5	$4.60 \pm 0.02$ ms	$4.69 \pm 0.01$ ms
10	$6.30 \pm 0.02$ ms	$6.49 \pm 0.02$ ms
50	$17.38 \pm 0.06$ ms	$18.15 \pm 0.03$ ms
100	$27.00 \pm 0.13$ ms	$28.59 \pm 0.16$ ms
500	$75.76 \pm 0.26$ ms	$81.68 \pm 0.23$ ms
1000	$128.97 \pm 0.36$ ms	$142.07 \pm 0.37$ ms
5000	$469.12 \pm 0.95$ ms	$504.99 \pm 1.03$ ms
10 000	$906.15 \pm 1.51$ ms	$985.19 \pm 1.77$ ms

(b) BigNumber and FixedPoint extension (left) and BigNumber, FixedPoint and Negative extensions.

*Table 7.4:* Mean and standard error of share times for participants when varying the input size  $\ell$  and the applied Jiff extensions. The extensions BigNumber, FixedPoint, and Negative are applied progressively from left to right and top to bottom.





## DISCUSSION

---

In this Section, we discuss the system proposed in Section 5.3 considering the privacy guarantees it provides and its performance. In Section 8.1, we discuss our results in terms of performance presented in Section 7.2 and revisit the research questions posed in Chapter 1. In Section 8.2, we discuss the limitations of our approach, both of technical and non-technical nature, and outline possible future work.

### 8.1 FINDINGS

Research question 1 is concerned with the security benefits of combining a TEE with SMPC as this work proposes. As noted in Section 5.3, our platform exceeds the security guarantees of both PeQES-like systems (relying solely on TEEs for providing security) and systems that ensure security through SMPC only. It is superior to the former by ensuring that even in the event of vulnerabilities in SGX or insecurely programmed platform code, the participants' privacy is not compromised. It improves on purely SMPC-based platforms by making hostile takeover of computing parties more difficult even for attackers that enjoy privileged access to the employed computing infrastructure. Further, it introduces no structural vulnerabilities: every attack possible on the proposed system is either also possible on platforms solely based on TEEs or purely SMPC-based ones. Unforeseen interactions between SGX and SMPC that introduce novel vulnerabilities are unlikely, as the two technologies operate on different levels of our system and are oblivious of one another.

Prompted through research question 2, we implemented four prototypes of platforms for conducting empirical studies in order to study their efficiency. We find that the platform we propose performs worse than every investigated alternative. This is evident through the results provided in Subsection 7.2.1, which show that the platform needs  $\approx 10$  min for performing one single t-test. That being said, it might still be *fast enough* for its intended purpose, which is running a script for statistical analysis of an empirical study exactly once. In light of the fact that empirical studies typically require months of preparation, study design and other work, up to several days of compute time for statistical analysis may be acceptable. However, in order to assess our prototype's fitness for this task, a more realistic statistical analysis script is needed for benchmarking, ideally as a result of cooperation with domain experts. We defer further experimentation of this kind to future work.

We also note that the low performance of our prototype might in part be explained not by conceptual issues, but by implementation-specific issues. Our prototype SMPC-SGX encompasses a large codebase. For example, it depends on a fully-fledged JavaScript runtime and on Jiff, which has 502 transitive dependencies. This leads to high memory consumption, which slows down execution. An optimisation of SMPC-SGX encompassing a switch to a library with lower memory consumption could improve performance. This improvement would be most effective if the JavaScript programming language were ditched in favour of a language not requiring a large runtime environment. Lastly, implementing

## DISCUSSION

Children	Self	Command	Shared	Object	Symbol
66.25%	0.00%	pal-sgx	libsysdb.so	[.]	libos_syscall_entry
66.25%	0.00%	pal-sgx	libsysdb.so	[.]	libos_emulate_syscall
47.85%	0.00%	pal-sgx	libsysdb.so	[.]	morestack
47.84%	17.74%	pal-sgx	libpal.so	[.]	ocall_gettime
34.14%	0.00%	pal-sgx	libsysdb.so	[.]	libos_syscall_clock_gettime
13.66%	0.00%	pal-sgx	libsysdb.so	[.]	libos_syscall_time
9.41%	0.00%	pal-sgx	[unknown]	[.]	0x0000000000000001
9.40%	9.40%	pal-sgx	libpal.so	[.]	ocall_send
9.40%	0.00%	pal-sgx	libsysdb.so	[.]	do_sendmsg
9.40%	0.00%	pal-sgx	libsysdb.so	[.]	send
9.40%	0.00%	pal-sgx	libpal.so	[.]	send
9.40%	0.00%	pal-sgx	libsysdb.so	[.]	libos_syscall_writev
6.79%	6.79%	pal-sgx	libpal.so	[.]	ocall_futex
6.76%	0.00%	pal-sgx	libsysdb.so	[.]	libos_syscall_futex
6.76%	0.00%	pal-sgx	libsysdb.so	[.]	_libos_syscall_futex (inlined)
3.88%	0.00%	pal-sgx	[unknown]	[.]	0xfffffffffffffff
3.43%	0.00%	pal-sgx	libsysdb.so	[.]	futex_wait (inlined)
3.40%	0.00%	pal-sgx	libpal.so	[.]	_PalEventSet
3.38%	0.00%	pal-sgx	libpal.so	[.]	_PalEventWait
3.34%	0.00%	pal-sgx	libsysdb.so	[.]	thread_wait (inlined)
3.33%	0.00%	pal-sgx	libsysdb.so	[.]	futex_wake
3.33%	0.00%	pal-sgx	libsysdb.so	[.]	wake_queue (inlined)
3.33%	0.00%	pal-sgx	libsysdb.so	[.]	thread_wakeup (inlined)
0.92%	0.92%	pal-sgx	libpal.so	[.]	ocall_poll

Figure 8.1: Results of profiling SMPC-SGX. Serving the system call `clock_gettime` accounts for 47.84% of time spent.

the prototype as a native SGX application as opposed to using Gramine-SGX could further improve performance, because non-critical parts of the application could be shifted outside the enclave.

Further, our evaluation was performed using an Intel processor preceding the IceLake architecture. Such older processors severely limit SGX’ performance through the following factors. As discussed in Section 2.3.1, the EPC of pre-IceLake processors is limited to 128 to 256 MB, which our application exceeds by far. This leads to significant overhead through page swapping. Further, pre-IceLake architectures forbid usage of certain instructions in enclave mode<sup>1</sup>, most importantly the RDTSC instruction. RDTSC is used for determining the current time, e. g., when application code calls the POSIX function `gettimeofday`<sup>2</sup>. During runtime, Gramine’s library OS detects whether or not RDTSC inside enclaves is supported on the current platform. If this is not the case, it implements `gettimeofday` through a more expensive call to non-enclave code, in particular `clock_gettime`<sup>3</sup>. This mechanism incurs a large overhead on applications making extensive use of `gettimeofday`. Through profiling (see Figure 8.1) of SMPC-SGX we discover that indeed much of its time running is spent serving the system call `clock_gettime`, which means that the application performs many calls to `gettimeofday`. However, we do not know the reason for this behaviour. The aforementioned complexity and multiple layers of abstraction found in SMPC-SGX make investigation hard.

Again, these issues are specific to older Intel architectures. It may thus well be the case that simply employing newer server hardware significantly improves performance up to the point of it being on par with SMPC-only.

The explanations for the poor performance of SMPC-SGX partly answer

- 1 See: <https://gramine.readthedocs.io/en/v1.4/performance.html#effects-of-system-calls-ocalls>
- 2 See: <https://man7.org/linux/man-pages/man2/gettimeofday.2.html>
- 3 See: [https://github.com/gramineproject/gramine/blob/v1.4/pal/src/host/linux/pal\\_misc.c#L32](https://github.com/gramineproject/gramine/blob/v1.4/pal/src/host/linux/pal_misc.c#L32)

research question 3, which asks for suitable technology for implementing the proposed platform. In principle, the technologies utilised in our prototypes can be used to build a platform for securely conducting empirical studies. Intel SGX is suitable as the utilised TEE because it is widely available. Gramine-SGX is, in principle, suitable because it enables unmodified language interpreters to enjoy SGX' security guarantees, allowing usage of high-level scripting languages for performing statistical analyses protected by enclaves. Jiff is suitable because it enables SMPC with security against a semi-honest minority on both client- and server-side. However, this software stack in combination with the hardware our experiments are performed on yields poor performance in executing the script for statistical analysis. On the basis of our experiments we cannot conclusively determine whether or not SMPC-SGX' performance suffices to be deployed in a real-world setting. Further experimentation, preferably using a realistic script for statistical analysis, is needed to come to such a conclusion. However, as the constraints on computation time in real-world empirical studies typically permit runtimes of several hours or days, we strongly believe that the proposed system is generally suitable for its envisioned tasks.

If this is not the case and compute time turns out to be a serious bottleneck of the platform, further possibilities for improving performance can be explored. For example, a conceivable optimisation is that the platform only protects computations via SMPC and TEEs that process sensitive data. Once data are sufficiently aggregated as to not threaten the privacy of participants<sup>4</sup>, analysis continues as a local computation protected by a TEE. This way, personal data of participants enjoy the enhanced security guarantees of our platform. Data that do not pose a privacy risk can be processed without SMPC-induced performance limitations and continue to have their integrity protected by the respective TEE.

#### 8.1.1 IMPACT OF THE NUMBER OF PARTICIPANTS

The results of experiment E1 presented in Subsection 7.2.1 suggest that the number of study participants  $m$  does not necessarily have a strong influence on compute time in SMPC-only and SMPC-SGX. However, this cannot lead to the conclusion that  $m$  never heavily impacts compute time. The script for statistical analysis used in experiment E1, which computes a two-sample t-test, requires a constant number of messages with regards to  $m$ . This is made possible by manually optimising the computation, which we discuss in Section 7.1. However, not all computations can be optimised as to have linear complexity in terms of communication. Computations that cause the exchange of a larger number of messages as  $m$  increases (e. g., by calculating the product of the input values) will indeed suffer from degraded performance. This is a fundamental difference of the SMPC-based and the non-SMPC-based platforms, which perform all computations locally and have negligibly short compute times even in the presence of computations requiring a large number of multiplications. Further research is needed to assess the potential impact of this issue in real-world settings.

<sup>4</sup> This can be measured through a suitable quantification of privacy, e. g., differential privacy [40].

## DISCUSSION

### 8.1.2 FEASIBILITY OF PREPROCESSING

Experiment E2 has investigated compute times in SMPC-only and SMPC-SGX when preprocessing is used. The time cost of preprocessing itself is no relevant factor in a real-world setting, as it can be performed in parallel to the data collection phase, in which participants are allowed to submit responses and which typically lasts days or weeks.

On the one hand, SMPC-only demonstrates that preprocessing in Jiff can reduce computation time by a significant amount. On the other hand, SMPC-SGX does not profit from preprocessing and performs much worse compared to experiments in which no preprocessing is used. We believe that this is due to the hardware limitations of pre-IceLake Intel processor architectures discussed in Subsection 7.2.2. On more recent architectures, SMPC-SGX should see similar efficiency gains in the compute phase as SMPC-only when preprocessing is used.

On a different note, preprocessing can negatively impact usability. Some frameworks for SMPC (e. g., Jiff) require script developers to explicitly specify the amount of preprocessing to be performed. For example, in order to enable preprocessing for the example script for statistical computation, we had to explicitly specify that we plan to perform three secret multiplications, one secret division and six divisions by a public constant. This reduces code maintainability and presumably acceptance of the platform. Future instances of the proposed platform should thus make use SMPC implementations that determine the necessary amount of preprocessing automatically (e. g., MP-SPDZ [61]).

### 8.1.3 IMPACT OF THE WORKLOAD SIZE

Experiment E3's results (presented in Subsection 7.2.3) show that compute time scales approximately proportionally with workload in SMPC-only. Naive and SGX-only show no such behaviour, which is probably due to optimisations performed by node.js at runtime as well as measurement errors as a result of the extremely short timespans being measured. SMPC-SGX's compute time seems to increase faster than the workload size, however, this is probably mostly due to increased page swapping and might not be present on Intel's IceLake and post-IceLake architectures.

### 8.1.4 IMPACT OF THE INPUT LENGTH

In experiment E4 we measure share time in participants. This metric is important for usability from the participants' point of view, since sharing is performed on their personal device. The results are presented in Subsection 7.2.4. Even in the event that a study requires participants to input 10 000 data points, share time remains under one second, which is noticeable for users, but acceptable. In the event that merely 500 inputs are required, the delay for sharing is  $< 0.1$  s, which does not impede user experience at all [89]. However, we note that sharing performance in the field may differ, as participants typically do not complete studies on potent server hardware. Surveys are rather answered on desktop computers or mobile devices. However, this does not call into question the applicability of our approach, as even very long share times can be worked

around, making impact on user experience barely noticeable. For example, as discussed in Subsection 8.2.1, sharing can be performed incrementally while the participant works on answering the survey, minimising the computations required at submission time.

#### 8.1.5 PERFORMANCE COMPARISON TO RELATED WORK

In this Section, we discuss SMPC-SGX' performance when compared to approaches in related work.

PeQES [81] is evaluated by performing two-sample t-tests on the participants' responses. It requires  $\approx 100$  ms/130 ms/900 ms for computing five t-tests on a sample size of 10/100/1000 survey responses, which is comparable to the performance of SGX-only. It comes as not surprise that it significantly outperforms SMPC-SGX.

For evaluating STAR [103], Servan-Schreiber et al. compute a two-sample t-test on a dataset consisting of 1000 data points, which takes  $\approx 90$  s. Notably, they argue that computation times ranging up to several minutes per statistical computation are acceptable in a real-world setting, as scientists typically do not require many of them. This boosts our confidence in the belief that SMPC-SGX, even with its current performance, is efficient enough for real-world deployment.

Bogdanov et al. evaluate Rmind [23] by performing various statistical tests, one of which is the two-sample t-test. Using three computation parties and a sample size of 2000, this operation takes 570 ms in the worst case. This significantly outperforms SMPC-SGX, which is expected. It also outperforms SMPC-only, which is likely due to the fact that Jiff uses less efficient protocols for basic arithmetic operations than ShareMind, which Rmind is based on. In particular, ShareMind uses highly optimised algorithms for division by public constant and for division of two secret values [22]. These two operations alone make up  $\approx 90\%$  of computation time in SMPC-only.

The framework by Chida et al. [29] for SMPC-based privacy-preserving statistical analyses outperforms SMPC-SGX, too. They are able to run two-sample t-tests using three computation parties in 1.64 s, however they do not specify the exact number of samples used as input.

It is evident that existing SMPC-based frameworks for statistical analysis achieve much better performance than SMPC-SGX. This is expected, as SMPC-SGX is the only approach in the comparison that employs a TEE. However, we see that existing frameworks also severely outperform SMPC-only. This can be due to a number of factors, most importantly the facts that existing frameworks for SMPC-based statistical analysis use more efficient algorithms for basic arithmetic operations than Jiff, use specialised algorithms for performing statistical tests and have been heavily optimised. This indicates that the performance we measured for SMPC-SGX is an upper bound for the potential performance of our general approach. Future work employing SMPC frameworks specialised on statistical analysis will presumably reduce compute times drastically.

## DISCUSSION

### 8.2 LIMITATIONS & FUTURE WORK

In this Section, we discuss limitations of this work, prompting further research. Subsection 8.2.1 discusses threats to the validity of our performance measurements. In Subsection 8.2.2, we discuss general limitations of our approach for realising a platform for securely conducting empirical studies and possible ways how to overcome them.

#### 8.2.1 EXPERIMENTAL APPARATUS

As discussed in the previous Sections, our experimental evaluation of the proposed system’s performance is lacking in some aspects. The results acquired for `SGX-only` and `SGX-SMPC` suffer from the limitations of the SGX implementation found on pre-IceLake Intel processor architectures. For `SGX-only`, this does not alter runtimes heavily as the prototype’s working data presumably fits into the enclave’s EPC. However, `SGX-SMPC`’s performance is impeded, both by the need to perform page swapping and the expensive workarounds needed to avoid unsupported instructions (e. g., `RDTSC`). Our experiments should thus be interpreted as providing an upper bound for the compute time of our approach. We assume that compute time on Intel IceLake or later architectures is much lower and in the same order of magnitude as `SMPC-only`’s.

However, `SMPC-only`’s real-world performance is presumably lower than under our experimental conditions. The reason for this is that we host all computation parties on one physical machine. Communication thus suffers no network delay, as would be case in a distributed system. As discussed in Section 4.1, network latency is a major reason for poor performance of SMPCs. Because computation of one two-sample t-test requires 280 548 messages in our evaluation, we believe that the real-world performance of `SMPC-only` may be significantly lower as the performance reported in Subsection 7.2.1.

Further, as discussed in Subsection 8.1.4, our measurements for share time may not be representative for share times imposed on real-world participants. We perform our experiments on server hardware, whereas participants typically complete studies through less potent desktop computers or mobile devices. However, this does not influence the applicability of the platform we propose. First, we assume that splitting survey responses on consumer-grade hardware is still fast enough to be practical, especially in light of the fact that the number of data points collected in a typical study is significantly lower than the number of data points shared in experiment E4. Second, the process of sharing responses can be optimised, e. g., by not performing the sharing after the survey has been completed, but incrementally as the participant answers survey questions. In essence, we highly doubt that sharing time impedes usability of the proposed platform.

The same is true for open time (results presented in Subsection 7.2.1). We measure open time in `SMPC-only` for one secret-shared value consistently as being under 10 ms. Note that this value includes communication cost for disseminating secret shares amongst the computation parties. In a real-world setting, the researcher performs the opening of secret values locally. Even in the case that this step is performed on weak hardware and for a large number of secret-shared values, we assume the computational overhead to be barely noticeable.

A different limitation of our experimental apparatus lies in the fact that the example script used in our experiments may not be representative of real-world statistical analyses in surveys. The script performs a two-sample t-test, which is commonly employed in empirical studies. However, performance measurements using a real-world study, including a real-world script for statistical analysis, would give more insights into the potential real-world performance of our approach.

We note that each of these limitations in our experimental apparatus can be addressed in future work. This would give further insights into the feasibility and potential performance of our approach. However, through this work we established a solid lower bound on our platform’s computational performance, indicating that it is generally suitable for conducting empirical studies.

### 8.2.2 APPROACH

The proposed platform aims to guarantee participant data in empirical studies. A prerequisite for achieving this goal is that the platform is actually used in such studies. However, we currently see one major obstacle for widespread adoption of our approach, which is the usability of the system. From a participant’s point of view, usability remains unchanged compared to other methods for conducting studies: surveys as still delivered as web applications, which the participant completes via their web browser. Verification of the ethics board’s signature as well as secret sharing can be performed in the background without requiring the participant to intervene. However, usability from the point of view of the researcher is impeded. Compared to an approach like PeQES [81] that requires preregistration of the script for statistical data analysis, computations now have to be defined as SMPCs. In the prototypes SMPC-only and SMPC-SGX, computations are expressed through the DSL provided by Jiff (see Subsection 6.2.1). However, using this DSL requires expert knowledge both on SMPC and JavaScript, which many researchers conducting empirical studies may not possess. For this reason, we recommend that the platform be programmable through a DSL inspired by a language commonly used for statistical analysis (e. g., Python, R) that hides all SMPC-specific details. For example, Rmind [23] and the statistics environment proposed by Chida et al. [29] are programmed in a DSL closely resembling R and can presumably be used without requiring expert knowledge on SMPC. If a similar approach were used in the proposed platform, the barrier for adoption may be lowered. Unfortunately, the source code of these statistics environments is not publicly available.

A different limitation concerning usability that may inhibit adoption of our approach is that interactive data analysis is no longer possible. Researchers must provide a script for statistical data analysis before data are collected. After approval by the ethics board, this script can no longer be modified. PeQES faces the same issue, and Meißner et al. [81] argue that researchers’ inability to perform interactive data exploration ensures rigorous scientific conduct and increases trust in the outcome of studies. They suggest that scripts for statistical data analysis may be designed and debugged with the help of pilot studies (which do not use on PeQES) prior to the main study or by running on synthetically generated survey responses. Through such techniques, researchers continue to be able to design the script for statistical analysis and form ideas for hypotheses

in an iterative process. Trust in scientific insights is established afterwards, when larger confirmatory studies are run through the platform.

A different approach to tackle this issue is given in Rmind [23]. In Rmind, a *study plan* specifies allowed and disallowed operations on the analysed data. Query restrictions are enforced by deleting the algorithms supporting disallowed operations from the computation parties. Further, the authors note that in principle, Rmind could ensure output privacy of calculations by employing *differentially private* [40] algorithms. These two techniques ensure that no private information of data subjects can be output via statistical analyses, even when interactive analysis by researchers is permitted. A similar approach could be utilised in a platform for securely conducting empirical studies: instead of signing the researcher's script for statistical analysis, the ethics board could provide a study plan specifying the level of privacy that must be ensured throughout the analysis using some suitable quantification of privacy (e. g., differential privacy [40]). The platform could enforce this level of privacy throughout interactive analysis. We stress that this approach does not necessarily prevent scientific misconduct through misuse of statistical methods, but ensures participants' privacy.

A different limitation of our platform lies in the fact that our approach does not protect metadata of participation in empirical studies. Meißner et al. [81] note that PeQES cannot prevent the platform provider from learning which IP addresses participated in a given study. The same is true for our platform.

We further note that the platform we propose is intended for conducting *quantitative* studies, i. e., studies in which responses consist of numerical values. We consider *qualitative* empirical studies out of scope for this work, however, future work should investigate possibilities for performing computation not only on numerical values, but also on text. Recent previous work has shown that such computations are possible and feasible through secret sharing-based SMPC protocols [93, 63, 94].



## SUMMARY & CONCLUSION

---

### 9.1 SUMMARY

In this Section, we summarise the contents of this work. In Chapter 1, we motivate our research and state its goals. We further pose research questions, which are investigated over the course of this work. Chapter 2 introduces relevant mathematical and technological concepts, in particular the concepts of secret sharing, SMPC and TEEs. In Chapter 3, we present previous work that facilitates usages of these concepts in a practical setting. Previous work is discussed in Chapter 4. We provide an overview on work aimed at combining SMPC and TEEs in Section 4.1 and on work that improves confidentiality in empirical studies or data analysis through either SMPC or TEEs in Section 4.2.

Chapter 5 describes our approach for strengthening security in empirical studies. Sections 5.1 to 5.2 discuss platforms that ensure privacy in empirical studies by utilising either TEEs or SMPC. Based on these protocols, Section 5.3 contains our main contribution, which is a platform fulfilling the same purpose, but providing stronger privacy guarantees for participants. We implement four prototypes realising such platforms using all possible combinations of using or not using TEEs and SMPC for strengthening privacy. The choice of technology for these prototypes is explained in Chapter 6. Chapter 7 describes the experimental setup and the experiments for evaluating the prototypes' performance. Further, it presents the results of these experiments. In Chapter 8, we discuss our approach in light of the security guarantees it provides and its performance. Further, we discuss the limitations of our approach and make suggestions for future work.

### 9.2 CONCLUSION

In this work, we propose a platform for conducting quantitative empirical studies while upholding scientific integrity and the privacy of participants. For achieving this goal, we combine the well-known techniques of SMPC and TEEs. We conclude that the proposed platform is indeed suitable for performing the desired task. It provides additional confidentiality guarantees compared to alternative approaches, however, these additional guarantees are bought at the price of prolonged computation time. The computational overhead imposed on survey participants is negligible. We argue that the proposed platform is suitable for use in empirical studies. However, further research is needed for determining the platform's real-world performance as well as its acceptance in the scientific community.



## BIBLIOGRAPHY

- 
- [1] Kinan Dak Albab. *JIFF: JavaScript Implementation of Federated Functionality*. Apr. 2019. URL: <https://www.youtube.com/watch?v=S-Iky0EgrfI> (visited on 08/03/2023).
  - [2] Kinan Dak Albab et al. 'Tutorial: Deploying Secure Multi-Party Computation on the Web Using JIFF'. In: *2019 IEEE Cybersecurity Development (SecDev)*. Sept. 2019, pp. 3–3. DOI: [10.1109/SecDev.2019.00013](https://doi.org/10.1109/SecDev.2019.00013).
  - [3] Tiago Alves and Don Felton. 'Trustzone : Integrated hardware and software security'. In: *Information Quarterly* 3 (2004), pp. 18–24. URL: <http://cir.nii.ac.jp/crid/1572824500864199424>.
  - [4] AMD. *AMD64 Architecture Programmer's Manual, Volume 2: System Programming, 24593*. en. 2006. URL: <https://www.amd.com/system/files/TechDocs/24593.pdf>.
  - [5] Ittai Anati et al. *Innovative Technology for CPU Based Attestation and Sealing*. Tech. rep. Aug. 2013. URL: <https://www.intel.com/content/www/us/en/developer/articles/technical/innovative-technology-for-cpu-based-attestation-and-sealing.html>.
  - [6] Ittai Anati et al. 'Inside 6th gen Intel® Core™: New microarchitecture code named skylake'. In: *2016 IEEE Hot Chips 28 Symposium (HCS)*. Aug. 2016, pp. 1–39. DOI: [10.1109/HOTCHIPS.2016.7936222](https://doi.org/10.1109/HOTCHIPS.2016.7936222).
  - [7] Ross Anderson. 'Cryptography and Competition Policy - Issues with "Trusted Computing"'. en. In: *Economics of Information Security*. Ed. by L. Jean Camp and Stephen Lewis. Advances in Information Security. Boston, MA: Springer US, 2004, pp. 35–52. ISBN: 978-1-4020-8090-6. DOI: [10.1007/1-4020-8090-5\\_3](https://doi.org/10.1007/1-4020-8090-5_3). URL: [https://doi.org/10.1007/1-4020-8090-5\\_3](https://doi.org/10.1007/1-4020-8090-5_3) (visited on 02/03/2023).
  - [8] Robin Ankele and Andrew Simpson. 'On the Performance of a Trustworthy Remote Entity in Comparison to Secure Multi-party Computation'. In: *2017 IEEE Trustcom/BigDataSE/ICCESS*. ISSN: 2324-9013. Aug. 2017, pp. 1115–1122. DOI: [10.1109/Trustcom/BigDataSE/ICCESS.2017.361](https://doi.org/10.1109/Trustcom/BigDataSE/ICCESS.2017.361).
  - [9] Robin Ankele et al. 'Applying the Trustworthy Remote Entity to Privacy-Preserving Multiparty Computation: Requirements and Criteria for Large-Scale Applications'. In: *2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)*. July 2016, pp. 414–422. DOI: [10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0077](https://doi.org/10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0077).
  - [10] Gilad Asharov and Yehuda Lindell. 'A Full Proof of the BGW Protocol for Perfectly Secure Multiparty Computation'. en. In: *Journal of Cryptology* 30.1 (Jan. 2017), pp. 58–151. ISSN: 1432-1378. DOI: [10.1007/s00145-015-9214-4](https://doi.org/10.1007/s00145-015-9214-4). URL: <https://doi.org/10.1007/s00145-015-9214-4> (visited on 27/03/2023).

## BIBLIOGRAPHY

- [11] Yonatan Aumann and Yehuda Lindell. ‘Security Against Covert Adversaries: Efficient Protocols for Realistic Adversaries’. en. In: *Journal of Cryptology* 23.2 (Apr. 2010), pp. 281–343. ISSN: 1432-1378. DOI: [10.1007/s00145-009-9040-7](https://doi.org/10.1007/s00145-009-9040-7). URL: <https://doi.org/10.1007/s00145-009-9040-7> (visited on 05/12/2022).
- [12] Marshall Ball, Tal Malkin and Mike Rosulek. *Garbling Gadgets for Boolean and Arithmetic Circuits*. Report Number: 969. 2016. URL: <https://eprint.iacr.org/2016/969> (visited on 21/03/2023).
- [13] Andrew Baumann, Marcus Peinado and Galen Hunt. ‘Shielding Applications from an Untrusted Cloud with Haven’. In: *ACM Transactions on Computer Systems* 33.3 (Aug. 2015), 8:1–8:26. ISSN: 0734-2071. DOI: [10.1145/2799647](https://doi.org/10.1145/2799647). URL: <https://doi.org/10.1145/2799647> (visited on 16/03/2023).
- [14] Donald Beaver. ‘Efficient Multiparty Protocols Using Circuit Randomization’. en. In: *Advances in Cryptology — CRYPTO ’91*. Ed. by Joan Feigenbaum. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1992, pp. 420–432. ISBN: 978-3-540-46766-3. DOI: [10.1007/3-540-46766-1\\_34](https://doi.org/10.1007/3-540-46766-1_34).
- [15] Donald Beaver. ‘Multiparty Protocols Tolerating Half Faulty Processors’. en. In: *Advances in Cryptology — CRYPTO’ 89 Proceedings*. Ed. by Gilles Brassard. Lecture Notes in Computer Science. New York, NY: Springer, 1990, pp. 560–572. ISBN: 978-0-387-34805-6. DOI: [10.1007/0-387-34805-0\\_49](https://doi.org/10.1007/0-387-34805-0_49).
- [16] Zuzana Beerliová-Trubíniová and Martin Hirt. ‘Perfectly-secure MPC with linear communication complexity’. In: *Theory of Cryptography: Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008. Proceedings 5*. Springer, 2008, pp. 213–230.
- [17] Michael Ben-Or, Shafi Goldwasser and Avi Wigderson. ‘Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation’. In: *Proceedings of the twentieth annual ACM symposium on Theory of computing*. STOC ’88. New York, NY, USA: Association for Computing Machinery, Jan. 1988, pp. 1–10. ISBN: 978-0-89791-264-8. DOI: [10.1145/62212.62213](https://doi.org/10.1145/62212.62213). URL: <https://doi.org/10.1145/62212.62213> (visited on 10/11/2022).
- [18] Azer Bestavros, Andrei Lapets and Mayank Varia. ‘User-centric distributed solutions for privacy-preserving analytics’. en. In: *Communications of the ACM* 60.2 (Jan. 2017), pp. 37–39. ISSN: 0001-0782, 1557-7317. DOI: [10.1145/3029603](https://doi.org/10.1145/3029603). URL: <https://dl.acm.org/doi/10.1145/3029603> (visited on 20/03/2023).
- [19] Yudhijit Bhattacharjee. ‘The Mind of a Con Man’. en-US. In: *The New York Times* (Apr. 2013). ISSN: 0362-4331. URL: <https://www.nytimes.com/2013/04/28/magazine/diederik-stapels-audacious-academic-fraud.html> (visited on 18/04/2023).

- [20] Dan Bogdanov, Sven Laur and Jan Willemson. ‘Sharemind: A Framework for Fast Privacy-Preserving Computations’. en. In: *Computer Security - ESORICS 2008*. Ed. by Sushil Jajodia and Javier Lopez. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2008, pp. 192–206. ISBN: 978-3-540-88313-5. DOI: [10.1007/978-3-540-88313-5\\_13](https://doi.org/10.1007/978-3-540-88313-5_13).
- [21] Dan Bogdanov, Riivo Talviste and Jan Willemson. ‘Deploying Secure Multi-Party Computation for Financial Data Analysis’. en. In: *Financial Cryptography and Data Security*. Ed. by Angelos D. Keromytis. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2012, pp. 57–64. ISBN: 978-3-642-32946-3. DOI: [10.1007/978-3-642-32946-3\\_5](https://doi.org/10.1007/978-3-642-32946-3_5).
- [22] Dan Bogdanov et al. ‘High-performance secure multi-party computation for data mining applications’. en. In: *International Journal of Information Security* 11.6 (Nov. 2012), pp. 403–418. ISSN: 1615-5270. DOI: [10.1007/s10207-012-0177-2](https://doi.org/10.1007/s10207-012-0177-2). URL: <https://doi.org/10.1007/s10207-012-0177-2> (visited on 26/04/2023).
- [23] Dan Bogdanov et al. ‘Rmind: A Tool for Cryptographically Secure Statistical Analysis’. In: *IEEE Transactions on Dependable and Secure Computing* 15.3 (May 2018). Conference Name: IEEE Transactions on Dependable and Secure Computing, pp. 481–495. ISSN: 1941-0018. DOI: [10.1109/TDSC.2016.2587623](https://doi.org/10.1109/TDSC.2016.2587623).
- [24] Christoph Bösch. ‘Privacy Engineering and Privacy Enhancing Technologies: S5 - Secure Multiparty Computation (MPC + Yao)’. Lecture Notes. Ulm University, 2021.
- [25] Ferdinand Brasser et al. ‘Software grand exposure: SGX cache attacks are practical’. In: *Proceedings of the 11th USENIX Conference on Offensive Technologies*. WOOT’17. USA: USENIX Association, Aug. 2017, p. 11. (Visited on 08/03/2023).
- [26] Jo Van Bulck et al. ‘Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution’. en. In: 2018, p. 991. ISBN: 978-1-939133-04-5. URL: <https://www.usenix.org/conference/usenixsecurity18/presentation/bulck> (visited on 03/04/2023).
- [27] Ran Canetti. ‘Security and Composition of Multiparty Cryptographic Protocols’. en. In: *Journal of Cryptology* 13.1 (Jan. 2000), pp. 143–202. ISSN: 1432-1378. DOI: [10.1007/s001459910006](https://doi.org/10.1007/s001459910006). URL: <https://doi.org/10.1007/s001459910006> (visited on 15/11/2022).
- [28] Yue Chen et al. *Downgrade Attack on TrustZone*. arXiv:1707.05082 [cs]. July 2017. DOI: [10.48550/arXiv.1707.05082](https://doi.org/10.48550/arXiv.1707.05082). URL: <http://arxiv.org/abs/1707.05082> (visited on 19/04/2023).
- [29] Koji Chida et al. ‘Implementation and evaluation of an efficient secure computation system using ‘R’ for healthcare statistics’. In: *Journal of the American Medical Informatics Association* 21.e2 (Oct. 2014), e326–e331. ISSN: 1067-5027. DOI: [10.1136/amiajnl-2014-002631](https://doi.org/10.1136/amiajnl-2014-002631). URL: <https://doi.org/10.1136/amiajnl-2014-002631> (visited on 23/03/2023).

## BIBLIOGRAPHY

- [30] Joseph I. Choi and Kevin R. B. Butler. ‘Secure Multiparty Computation and Trusted Hardware: Examining Adoption Challenges and Opportunities’. en. In: *Security and Communication Networks* 2019 (Apr. 2019). Publisher: Hindawi, e1368905. ISSN: 1939-0114. DOI: [10.1155/2019/1368905](https://doi.org/10.1155/2019/1368905). URL: <https://www.hindawi.com/journals/scn/2019/1368905/> (visited on 23/03/2023).
- [31] Benny Chor et al. ‘Verifiable secret sharing and achieving simultaneity in the presence of faults’. In: *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*. ISSN: 0272-5428. Oct. 1985, pp. 383–395. DOI: [10.1109/SFCS.1985.64](https://doi.org/10.1109/SFCS.1985.64).
- [32] Richard Erwin Cleve. ‘Limits on the security of coin flips when half the processors are faulty’. In: *Proceedings of the eighteenth annual ACM symposium on Theory of computing*. STOC ’86. New York, NY, USA: Association for Computing Machinery, Nov. 1986, pp. 364–369. ISBN: 978-0-89791-193-1. DOI: [10.1145/12130.12168](https://doi.org/10.1145/12130.12168). URL: <https://doi.org/10.1145/12130.12168> (visited on 17/11/2022).
- [33] Chris Clifton et al. ‘Tools for privacy preserving distributed data mining’. In: *ACM SIGKDD Explorations Newsletter* 4.2 (Dec. 2002), pp. 28–34. ISSN: 1931-0145. DOI: [10.1145/772862.772867](https://doi.org/10.1145/772862.772867). URL: <https://doi.org/10.1145/772862.772867> (visited on 20/03/2023).
- [34] Victor Costan and Srinivas Devadas. *Intel SGX Explained*. Report Number: o86. 2016. URL: <https://eprint.iacr.org/2016/086> (visited on 08/11/2022).
- [35] Victor Costan, Ilia Lebedev and Srinivas Devadas. ‘Sanctum: minimal hardware extensions for strong software isolation’. In: *Proceedings of the 25th USENIX Conference on Security Symposium*. SEC’16. USA: USENIX Association, Aug. 2016, pp. 857–874. ISBN: 978-1-931971-32-4. (Visited on 07/12/2022).
- [36] N. a. C. Cressie and H. J. Whitford. ‘How to Use the Two Sample t-Test’. en. In: *Biometrical Journal* 28.2 (1986), pp. 131–148. ISSN: 1521-4036. DOI: [10.1002/bimj.4710280202](https://doi.org/10.1002/bimj.4710280202). URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/bimj.4710280202> (visited on 11/04/2023).
- [37] Ivan Damgård and Jesper Buus Nielsen. ‘Scalable and Unconditionally Secure Multiparty Computation’. en. In: *Advances in Cryptology - CRYPTO 2007*. Ed. by Alfred Menezes. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2007, pp. 572–590. ISBN: 978-3-540-74143-5. DOI: [10.1007/978-3-540-74143-5\\_32](https://doi.org/10.1007/978-3-540-74143-5_32).
- [38] Ed Dawson and Diane Donovan. ‘The breadth of Shamir’s secret-sharing scheme’. en. In: *Computers & Security* 13.1 (Feb. 1994), pp. 69–78. ISSN: 0167-4048. DOI: [10.1016/0167-4048\(94\)90097-3](https://doi.org/10.1016/0167-4048(94)90097-3). URL: <https://www.sciencedirect.com/science/article/pii/0167404894900973> (visited on 22/11/2022).
- [39] Daniel Demmler, Thomas Schneider and Michael Zohner. ‘ABY-A framework for efficient mixed-protocol secure two-party computation.’ In: *NDSS*. 2015.

- [40] Cynthia Dwork. ‘Differential Privacy: A Survey of Results’. en. In: *Theory and Applications of Models of Computation*. Ed. by Manindra Agrawal et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2008, pp. 1–19. ISBN: 978-3-540-79228-4. DOI: [10.1007/978-3-540-79228-4\\_1](https://doi.org/10.1007/978-3-540-79228-4_1).
- [41] Khaled El Emam et al. ‘A Protocol for the Secure Linking of Registries for HPV Surveillance’. en. In: *PLOS ONE* 7.7 (July 2012). Publisher: Public Library of Science, e39915. ISSN: 1932-6203. DOI: [10.1371/journal.pone.0039915](https://doi.org/10.1371/journal.pone.0039915). URL: <https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0039915> (visited on 23/03/2023).
- [42] Benjamin Erb et al. *Emerging Privacy Issues in Times of Open Science*. en-us. June 2021. DOI: [10.31234/osf.io/u236e](https://doi.org/10.31234/osf.io/u236e). URL: <https://psyarxiv.com/u236e/> (visited on 18/10/2022).
- [43] David Evans, Vladimir Kolesnikov and Mike Rosulek. ‘A Pragmatic Introduction to Secure Multi-Party Computation’. In: *Foundations and Trends in Privacy and Security* 2.2-3 (Dec. 2018), pp. 70–246. ISSN: 2474-1558. DOI: [10.1561/33000000019](https://doi.org/10.1561/33000000019). URL: <https://doi.org/10.1561/33000000019> (visited on 02/11/2022).
- [44] Shufan Fei et al. ‘Security Vulnerabilities of SGX and Countermeasures: A Survey’. In: *ACM Computing Surveys* 54.6 (July 2021), 126:1–126:36. ISSN: 0360-0300. DOI: [10.1145/3456631](https://doi.org/10.1145/3456631). URL: <https://doi.org/10.1145/3456631> (visited on 08/03/2023).
- [45] Edward W. Felten. ‘Understanding trusted computing: will its benefits outweigh its drawbacks?’ In: *IEEE Security & Privacy* 1.3 (May 2003). Conference Name: IEEE Security & Privacy, pp. 60–62. ISSN: 1558-4046. DOI: [10.1109/MSECP.2003.1203224](https://doi.org/10.1109/MSECP.2003.1203224).
- [46] Oded Goldreich, Silvio Micali and Avi Wigderson. ‘How to play ANY mental game’. In: *Proceedings of the nineteenth annual ACM symposium on Theory of computing*. STOC ’87. New York, NY, USA: Association for Computing Machinery, Jan. 1987, pp. 218–229. ISBN: 978-0-89791-221-1. DOI: [10.1145/28395.28420](https://doi.org/10.1145/28395.28420). URL: <https://doi.org/10.1145/28395.28420> (visited on 10/11/2022).
- [47] Mario Gollwitzer et al. ‘Management und Bereitstellung von Forschungsdaten in der Psychologie: Überarbeitung der DGPs-Empfehlungen’. In: *Psychologische Rundschau* 72.2 (Apr. 2021). Publisher: Hogrefe Verlag, pp. 132–146. ISSN: 0033-3042. DOI: [10.1026/0033-3042/a000514](https://doi.org/10.1026/0033-3042/a000514). URL: <https://econtent.hogrefe.com/doi/full/10.1026/0033-3042/a000514> (visited on 19/04/2023).
- [48] Debayan Gupta et al. ‘Using Intel Software Guard Extensions for Efficient Two-Party Secure Function Evaluation’. en. In: *Financial Cryptography and Data Security*. Ed. by Jeremy Clark et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2016, pp. 302–318. ISBN: 978-3-662-53357-4. DOI: [10.1007/978-3-662-53357-4\\_20](https://doi.org/10.1007/978-3-662-53357-4_20).

## BIBLIOGRAPHY

- [49] John L. Gustafson. 'Moore's Law'. en. In: *Encyclopedia of Parallel Computing*. Ed. by David Padua. Boston, MA: Springer US, 2011, pp. 1177–1184. ISBN: 978-0-387-09766-4. DOI: [10.1007/978-0-387-09766-4\\_81](https://doi.org/10.1007/978-0-387-09766-4_81). URL: [https://doi.org/10.1007/978-0-387-09766-4\\_81](https://doi.org/10.1007/978-0-387-09766-4_81) (visited on 23/03/2023).
- [50] Marcella Hastings et al. 'SoK: General Purpose Compilers for Secure Multi-Party Computation'. In: *2019 IEEE Symposium on Security and Privacy (SP)*. ISSN: 2375-1207. May 2019, pp. 1220–1237. DOI: [10.1109/SP.2019.00028](https://doi.org/10.1109/SP.2019.00028).
- [51] Megan L. Head et al. 'The Extent and Consequences of P-Hacking in Science'. en. In: *PLOS Biology* 13.3 (Mar. 2015). Publisher: Public Library of Science, e1002106. ISSN: 1545-7885. DOI: [10.1371/journal.pbio.1002106](https://doi.org/10.1371/journal.pbio.1002106). URL: <https://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.1002106> (visited on 18/04/2023).
- [52] Siam Hussain et al. 'TinyGarble2: Smart, Efficient, and Scalable Yao's Garble Circuit'. In: *Proceedings of the 2020 Workshop on Privacy-Preserving Machine Learning in Practice*. PPMLP'20. New York, NY, USA: Association for Computing Machinery, Nov. 2020, pp. 65–67. ISBN: 978-1-4503-8088-1. DOI: [10.1145/3411501.3419433](https://doi.org/10.1145/3411501.3419433). URL: <https://doi.org/10.1145/3411501.3419433> (visited on 21/03/2023).
- [53] Intel Corporation. *Intel® Software Guard Extensions (Intel® SGX)*. Tech. rep. Reference Number: 332680-002. June 2015. URL: <https://www.intel.com/content/dam/develop/external/us/en/documents/332680-002-621824.pdf>.
- [54] *Intel® 64 and IA-32 Architectures Software Developer Manuals*. en. URL: <https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html> (visited on 02/03/2023).
- [55] *Intel® Software Guard Extensions*. en. URL: <https://software.intel.com/sgx> (visited on 08/12/2022).
- [56] International Statistical Institute. 'Declaration on professional ethics'. In: International Statistical Institute Voorburg, 1985. URL: <https://www.isi-web.org/isi-declaration-professional-ethics-1985>.
- [57] John P. A. Ioannidis. 'Why Most Published Research Findings Are False'. en. In: *PLOS Medicine* 2.8 (Aug. 2005). Publisher: Public Library of Science, e124. ISSN: 1549-1676. DOI: [10.1371/journal.pmed.0020124](https://doi.org/10.1371/journal.pmed.0020124). URL: <https://journals.plos.org/plosmedicine/article?id=10.1371/journal.pmed.0020124> (visited on 18/04/2023).
- [58] Ayman Jarrous and Benny Pinkas. 'Canon-MPC, A System for Casual Non-Interactive Secure Multi-Party Computation Using Native Client'. In: *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*. WPES '13. New York, NY, USA: Association for Computing Machinery, Nov. 2013, pp. 155–166. ISBN: 978-1-4503-2485-4. DOI: [10.1145/2517840.2517845](https://doi.org/10.1145/2517840.2517845). URL: <https://doi.org/10.1145/2517840.2517845> (visited on 09/03/2023).



- [59] Liina Kamm et al. ‘A new way to protect privacy in large-scale genome-wide association studies’. In: *Bioinformatics* 29.7 (Apr. 2013), pp. 886–893. ISSN: 1367-4803. DOI: [10.1093/bioinformatics/btt066](https://doi.org/10.1093/bioinformatics/btt066). URL: <https://doi.org/10.1093/bioinformatics/btt066> (visited on 23/03/2023).
- [60] Ryan Karl et al. ‘Developing non-interactive MPC with trusted hardware for enhanced security’. en. In: *International Journal of Information Security* 21.4 (Aug. 2022), pp. 777–797. ISSN: 1615-5270. DOI: [10.1007/s10207-022-00583-w](https://doi.org/10.1007/s10207-022-00583-w). URL: <https://doi.org/10.1007/s10207-022-00583-w> (visited on 27/03/2023).
- [61] Marcel Keller. ‘MP-SPDZ: A Versatile Framework for Multi-Party Computation’. In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’20. New York, NY, USA: Association for Computing Machinery, Nov. 2020, pp. 1575–1590. ISBN: 978-1-4503-7089-9. DOI: [10.1145/3372297.3417872](https://doi.org/10.1145/3372297.3417872). URL: <https://doi.org/10.1145/3372297.3417872> (visited on 31/01/2023).
- [62] Norbert L. Kerr. ‘HARKing: Hypothesizing After the Results are Known’. en. In: *Personality and Social Psychology Review* 2.3 (Aug. 1998). Publisher: SAGE Publications Inc, pp. 196–217. ISSN: 1088-8683. DOI: [10.1207/s15327957pspr0203\\_4](https://doi.org/10.1207/s15327957pspr0203_4). URL: [https://doi.org/10.1207/s15327957pspr0203\\_4](https://doi.org/10.1207/s15327957pspr0203_4) (visited on 18/04/2023).
- [63] Brian Knott et al. ‘CrypTen: Secure Multi-Party Computation Meets Machine Learning’. In: *Advances in Neural Information Processing Systems*. Vol. 34. Curran Associates, Inc., 2021, pp. 4961–4973. URL: <https://proceedings.neurips.cc/paper/2021/hash/2754518221cfbc8d25c13a06a4cb8421-Abstract.html> (visited on 20/04/2023).
- [64] Toshiki Kobayashi et al. ‘SAFES: Sand-boxed Architecture for Frequent Environment Self-measurement’. In: *Proceedings of the 3rd Workshop on System Software for Trusted Execution*. SysTEX ’18. New York, NY, USA: Association for Computing Machinery, Jan. 2018, pp. 37–41. ISBN: 978-1-4503-5998-6. DOI: [10.1145/3268935.3268939](https://doi.org/10.1145/3268935.3268939). URL: <https://doi.org/10.1145/3268935.3268939> (visited on 02/03/2023).
- [65] Kubilay Ahmet Küçük et al. ‘Exploring the use of Intel SGX for Secure Many-Party Applications’. In: *Proceedings of the 1st Workshop on System Software for Trusted Execution*. SysTEX ’16. New York, NY, USA: Association for Computing Machinery, Dec. 2016, pp. 1–6. ISBN: 978-1-4503-4670-2. DOI: [10.1145/3007788.3007793](https://doi.org/10.1145/3007788.3007793). URL: <https://doi.org/10.1145/3007788.3007793> (visited on 23/10/2022).
- [66] Andrei Lapets et al. ‘Accessible Privacy-Preserving Web-Based Data Analysis for Assessing and Addressing Economic Inequalities’. In: *Proceedings of the 1st ACM SIGCAS Conference on Computing and Sustainable Societies*. COMPASS ’18. New York, NY, USA: Association for Computing Machinery, June 2018, pp. 1–5. ISBN: 978-1-4503-5816-3. DOI: [10.1145/3209811.3212701](https://doi.org/10.1145/3209811.3212701). URL: <https://doi.org/10.1145/3209811.3212701> (visited on 20/03/2023).

## BIBLIOGRAPHY

- [67] Andrei Lapets et al. *Role-Based Ecosystem for Design, Development, and Deployment of Secure Multi-Party Data Analytics Applications*. Report Number: 803. 2017. URL: <https://eprint.iacr.org/2017/803> (visited on 08/03/2023).
- [68] Andrei Lapets et al. 'Secure MPC for Analytics as a Web Application'. In: *2016 IEEE Cybersecurity Development (SecDev)*. Nov. 2016, pp. 73–74. DOI: [10.1109/SecDev.2016.027](https://doi.org/10.1109/SecDev.2016.027).
- [69] Andrei Lapets et al. *Secure multi-party computation for analytics deployed as a lightweight web application*. Technical Report. Accepted: 2017-04-26T18:43:47Z. Computer Science Department, Boston University, July 2016. URL: <https://open.bu.edu/handle/2144/21786> (visited on 20/03/2023).
- [70] Andrei Lapets et al. *Web-based multi-party computation with application to anonymous aggregate compensation analytics*. Technical Report. Accepted: 2017-04-26T18:43:32Z. Computer Science Department, Boston University, Aug. 2015. URL: <https://open.bu.edu/handle/2144/21773> (visited on 17/11/2022).
- [71] *LatticeX-Foundation/Rosetta*. original-date: 2020-04-08T14:25:50Z. Feb. 2023. URL: <https://github.com/LatticeX-Foundation/Rosetta> (visited on 21/03/2023).
- [72] John Launchbury et al. 'Efficient lookup-table protocol in secure multi-party computation'. In: *Proceedings of the 17th ACM SIGPLAN international conference on Functional programming*. ICFP '12. New York, NY, USA: Association for Computing Machinery, Sept. 2012, pp. 189–200. ISBN: 978-1-4503-1054-3. DOI: [10.1145/2364527.2364556](https://doi.org/10.1145/2364527.2364556). URL: <https://doi.org/10.1145/2364527.2364556> (visited on 21/03/2023).
- [73] Dayeol Lee et al. 'Keystone: an open framework for architecting trusted execution environments'. In: *Proceedings of the Fifteenth European Conference on Computer Systems*. EuroSys '20. New York, NY, USA: Association for Computing Machinery, Apr. 2020, pp. 1–16. ISBN: 978-1-4503-6882-7. DOI: [10.1145/3342195.3387532](https://doi.org/10.1145/3342195.3387532). URL: <https://doi.org/10.1145/3342195.3387532> (visited on 07/02/2023).
- [74] Mengyuan Li et al. 'CIPHERLEAKS: Breaking Constant-time Cryptography on AMD SEV via the Ciphertext Side Channel'. en. In: 2021, pp. 717–732. ISBN: 978-1-939133-24-3. URL: <https://www.usenix.org/conference/usenixsecurity21/presentation/li-mengyuan> (visited on 19/04/2023).
- [75] David Lie et al. 'Architectural support for copy and tamper resistant software'. In: *ACM SIGPLAN Notices* 35.11 (Nov. 2000), pp. 168–177. ISSN: 0362-1340. DOI: [10.1145/356989.357005](https://doi.org/10.1145/356989.357005). URL: <https://doi.org/10.1145/356989.357005> (visited on 07/02/2023).
- [76] Yehuda Lindell. *Secure Multiparty Computation (MPC)*. Report Number: 300. 2020. URL: <https://eprint.iacr.org/2020/300> (visited on 01/11/2022).

- [77] Moxie Marlinspike. *Technology preview: Private contact discovery for Signal*. en. Sept. 2017. URL: <https://signal.org/blog/private-contact-discovery/> (visited on 03/04/2023).
- [78] Andrew Martin. 'The ten-page introduction to Trusted Computing'. English. In: (2008). Edition: Author's Original Version Number: Author's Original. URL: <https://ora.ox.ac.uk/objects/uuid:a4a7ae67-7b2a-4516-801d-9379d613bab4> (visited on 02/03/2023).
- [79] Frank McKeen et al. 'Innovative instructions and software model for isolated execution'. In: *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*. HASP '13. New York, NY, USA: Association for Computing Machinery, June 2013, p. 1. ISBN: 978-1-4503-2118-1. DOI: 10.1145/2487726.2488368. URL: <https://doi.org/10.1145/2487726.2488368> (visited on 06/12/2022).
- [80] Frank McKeen et al. 'Intel® Software Guard Extensions (Intel® SGX) Support for Dynamic Memory Management Inside an Enclave'. In: *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016*. HASP 2016. New York, NY, USA: Association for Computing Machinery, June 2016, pp. 1–9. ISBN: 978-1-4503-4769-3. DOI: 10.1145/2948618.2954331. URL: <https://dl.acm.org/doi/10.1145/2948618.2954331> (visited on 11/04/2023).
- [81] Dominik Meißner et al. 'PeQES: a platform for privacy-enhanced quantitative empirical studies'. In: *Proceedings of the 36th Annual ACM Symposium on Applied Computing*. SAC '21. New York, NY, USA: Association for Computing Machinery, Mar. 2021, pp. 1226–1234. ISBN: 978-1-4503-8104-8. DOI: 10.1145/3412841.3441997. URL: <https://doi.org/10.1145/3412841.3441997> (visited on 19/10/2022).
- [82] Xinyuan Miao et al. *Lejacon: A Lightweight and Efficient Approach to Java Confidential Computing on SGX*. en. URL: [https://ddst.sjtu.edu.cn/Management/Upload/\[News\]a845acae286b470bb55013c1b5e425e2/20232101456536725sSV.pdf](https://ddst.sjtu.edu.cn/Management/Upload/[News]a845acae286b470bb55013c1b5e425e2/20232101456536725sSV.pdf).
- [83] Moose. original-date: 2020-04-22T12:16:15Z. Mar. 2023. URL: <https://github.com/tf-encrypted/moose> (visited on 21/03/2023).
- [84] Multiparty.org Development Team. *JavaScript Implementation of Federated Functionalities*. original-date: 2017-05-31T21:23:20Z. May 2018. URL: <https://github.com/multiparty/jiff> (visited on 08/03/2023).
- [85] Multiparty.org Development Team. *Secure MPC Protocol for (Sample Standard Deviation)*. original-date: 2017-05-31T21:23:20Z. Nov. 2020. URL: <https://github.com/multiparty/jiff/blob/8ea565d3d0becde8f71243fb9daea6ef0ba9bb7e/demos/standard-deviation/standdevprotocol.pdf> (visited on 11/04/2023).
- [86] Arvind Narayanan and Vitaly Shmatikov. 'Robust De-anonymization of Large Sparse Datasets'. In: *2008 IEEE Symposium on Security and Privacy (sp 2008)*. ISSN: 2375-1207. May 2008, pp. 111–125. DOI: 10.1109/SP.2008.33.

## BIBLIOGRAPHY

- [87] Arvind Narayanan et al. ‘On the Feasibility of Internet-Scale Author Identification’. In: *2012 IEEE Symposium on Security and Privacy*. ISSN: 2375-1207. May 2012, pp. 300–314. DOI: [10.1109/SP.2012.46](https://doi.org/10.1109/SP.2012.46).
- [88] Jakob Nielsen. *Nielsen’s Law of Internet Bandwidth*. en. Jan. 2023. URL: <https://www.nngroup.com/articles/law-of-bandwidth/> (visited on 23/03/2023).
- [89] Jakob Nielsen. *Response Time Limits*. en. Jan. 1993. URL: <https://www.nngroup.com/articles/response-times-3-important-limits/> (visited on 26/04/2023).
- [90] Open Science Collaboration. ‘Estimating the reproducibility of psychological science’. In: *Science* 349.6251 (Aug. 2015). Publisher: American Association for the Advancement of Science, aac4716. DOI: [10.1126/science.aac4716](https://doi.org/10.1126/science.aac4716). URL: <https://www.science.org/doi/10.1126/science.aac4716> (visited on 18/04/2023).
- [91] Bernardo Portela et al. ‘Secure Multiparty Computation from SGX’. In: *Financial Cryptography and Data Security 2017* (Apr. 2017). Publisher: International Financial Cryptography Association.
- [92] Tal Rabin and Michael Ben-Or. ‘Verifiable secret sharing and multiparty protocols with honest majority’. In: *Proceedings of the twenty-first annual ACM symposium on Theory of computing*. STOC ’89. New York, NY, USA: Association for Computing Machinery, Feb. 1989, pp. 73–85. ISBN: 978-0-89791-307-2. DOI: [10.1145/73007.73014](https://doi.org/10.1145/73007.73014). URL: <https://dl.acm.org/doi/10.1145/73007.73014> (visited on 27/03/2023).
- [93] Devin Reich et al. ‘Privacy-Preserving Classification of Personal Text Messages with Secure Multi-Party Computation’. In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc., 2019. URL: <https://proceedings.neurips.cc/paper/2019/hash/a501bebf79d570651ff601788ea9d16d-Abstract.html> (visited on 20/04/2023).
- [94] Amanda Resende et al. ‘Fast Privacy-Preserving Text Classification Based on Secure Multiparty Computation’. In: *IEEE Transactions on Information Forensics and Security* 17 (2022). Conference Name: IEEE Transactions on Information Forensics and Security, pp. 428–442. ISSN: 1556-6021. DOI: [10.1109/TIFS.2022.3144007](https://doi.org/10.1109/TIFS.2022.3144007).
- [95] Peter Rindal and Mike Rosulek. *Faster Malicious 2-party Secure Computation with Online/Offline Dual Execution*. Report Number: 632. 2016. URL: <https://eprint.iacr.org/2016/632> (visited on 21/03/2023).
- [96] *Rising to the Challenge — Data Security with Intel Confidential Computing*. en. Section: Security. Jan. 2022. URL: <https://community.intel.com/t5/Blogs/Products-and-Solutions/Security/Rising-to-the-Challenge-Data-Security-with-Intel-Confidential/post/1353141> (visited on 02/03/2023).

- [97] Luc Rocher, Julien M. Hendrickx and Yves-Alexandre de Montjoye. ‘Estimating the success of re-identifications in incomplete datasets using generative models’. en. In: *Nature Communications* 10.1 (July 2019). Number: 1 Publisher: Nature Publishing Group, p. 3069. ISSN: 2041-1723. DOI: [10.1038/s41467-019-10933-3](https://doi.org/10.1038/s41467-019-10933-3). URL: <https://www.nature.com/articles/s41467-019-10933-3/> (visited on 19/04/2023).
- [98] David Cerezo Sánchez. *Raziel: Private and Verifiable Smart Contracts on Blockchains*. Report Number: 878. 2017. URL: <https://eprint.iacr.org/2017/878> (visited on 21/03/2023).
- [99] SCALE and MAMBA. original-date: 2018-05-02T10:40:57Z. Mar. 2023. URL: <https://github.com/KULeuven-COSIC/SCALE-MAMBA> (visited on 21/03/2023).
- [100] Berry Schoenmakers. *MPyC Multiparty Computation in Python*. original-date: 2018-04-19T05:43:08Z. Mar. 2023. URL: <https://github.com/lischoe/mpyc> (visited on 16/03/2023).
- [101] Berry Schoenmakers. ‘MPyC—Python package for secure multiparty computation’. In: *Workshop on the Theory and Practice of MPC*. <https://github.com/lischoe/mpyc>. 2018.
- [102] Felix Schuster et al. ‘VC3: Trustworthy Data Analytics in the Cloud Using SGX’. In: *2015 IEEE Symposium on Security and Privacy*. ISSN: 2375-1207. May 2015, pp. 38–54. DOI: [10.1109/SP.2015.10](https://doi.org/10.1109/SP.2015.10).
- [103] Sacha Servan-Schreiber et al. *STAR: Statistical Tests with Auditable Results*. arXiv:1901.10875 [cs, stat]. Oct. 2019. DOI: [10.48550/arXiv.1901.10875](https://doi.org/10.48550/arXiv.1901.10875). URL: <http://arxiv.org/abs/1901.10875> (visited on 20/10/2022).
- [104] Adi Shamir. ‘How to share a secret’. In: *Communications of the ACM* 22.11 (Nov. 1979), pp. 612–613. ISSN: 0001-0782. DOI: [10.1145/359168.359176](https://doi.org/10.1145/359168.359176). URL: <https://doi.org/10.1145/359168.359176> (visited on 17/11/2022).
- [105] Fahad Shaon et al. ‘SGX-BigMatrix: A Practical Encrypted Data Analytic Framework With Trusted Processors’. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’17. New York, NY, USA: Association for Computing Machinery, Oct. 2017, pp. 1211–1228. ISBN: 978-1-4503-4946-8. DOI: [10.1145/3133956.3134095](https://doi.org/10.1145/3133956.3134095). URL: <https://doi.org/10.1145/3133956.3134095> (visited on 23/10/2022).
- [106] Patrick E. Shrout and Joseph L. Rodgers. ‘Psychology, Science, and Knowledge Construction: Broadening Perspectives from the Replication Crisis’. eng. In: *Annual Review of Psychology* 69 (Jan. 2018), pp. 487–510. ISSN: 1545-2085. DOI: [10.1146/annurev-psych-122216-011845](https://doi.org/10.1146/annurev-psych-122216-011845).
- [107] Joseph P. Simmons, Leif D. Nelson and Uri Simonsohn. ‘False-Positive Psychology: Undisclosed Flexibility in Data Collection and Analysis Allows Presenting Anything as Significant’. en. In: *Psychological Science* 22.11 (Nov. 2011). Publisher: SAGE Publications Inc, pp. 1359–1366. ISSN: 0956-7976. DOI: [10.1177/0956797611417632](https://doi.org/10.1177/0956797611417632). URL: <https://doi.org/10.1177/0956797611417632> (visited on 18/04/2023).

## BIBLIOGRAPHY

- [108] Ebrahim M. Songhori et al. 'TinyGarble: Highly Compressed and Scalable Sequential Garbled Circuits'. In: *2015 IEEE Symposium on Security and Privacy*. ISSN: 2375-1207. May 2015, pp. 411–428. DOI: [10.1109/SP.2015.32](https://doi.org/10.1109/SP.2015.32).
- [109] Richard Stallman. 'Can you trust your computer?' In: *NewsForge: The Online Newspaper for Linux and Open Source* October (2002).
- [110] Douglas R. Stinson. 'An Explication of Secret Sharing Schemes'. en. In: *Designs, Codes and Cryptography* 2.4 (Dec. 1992), pp. 357–390. ISSN: 1573-7586. DOI: [10.1007/BF00125203](https://doi.org/10.1007/BF00125203). URL: <https://doi.org/10.1007/BF00125203> (visited on 29/11/2022).
- [111] Latanya Sweeney. 'k-Anonymity: A Model for Protecting Privacy'. In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10.05 (Oct. 2002). Publisher: World Scientific Publishing Co., pp. 557–570. ISSN: 0218-4885. DOI: [10.1142/S0218488502001648](https://doi.org/10.1142/S0218488502001648). URL: <https://www.worldscientific.com/doi/abs/10.1142/S0218488502001648> (visited on 19/04/2023).
- [112] Trusted Computing Group. *Trusted platform module library specification (tpm2.0)*. 2013. URL: <https://trustedcomputinggroup.org/resource/tpm-library-specification/>.
- [113] Chia-Che Tsai, Donald E. Porter and Mona Vij. 'Graphene-SGX: a practical library OS for unmodified applications on SGX'. In: *Proceedings of the 2017 USENIX Conference on Usenix Annual Technical Conference*. USENIX ATC '17. USA: USENIX Association, July 2017, pp. 645–658. ISBN: 978-1-931971-38-6. (Visited on 16/01/2023).
- [114] Chia-Che Tsai et al. 'Cooperation and security isolation of library OSES for multi-process applications'. In: *Proceedings of the Ninth European Conference on Computer Systems*. EuroSys '14. New York, NY, USA: Association for Computing Machinery, Apr. 2014, pp. 1–14. ISBN: 978-1-4503-2704-6. DOI: [10.1145/2592798.2592812](https://doi.org/10.1145/2592798.2592812). URL: <https://doi.org/10.1145/2592798.2592812> (visited on 16/03/2023).
- [115] Stephan Van Schaik et al. *SGAxe: How SGX fails in practice*. 2020. URL: <https://sgaxe.com/files/SGAxe.pdf>.
- [116] Edward Vul et al. 'Puzzlingly High Correlations in fMRI Studies of Emotion, Personality, and Social Cognition'. en. In: *Perspectives on Psychological Science* 4.3 (May 2009). Publisher: SAGE Publications Inc, pp. 274–290. ISSN: 1745-6916. DOI: [10.1111/j.1745-6924.2009.01125.x](https://doi.org/10.1111/j.1745-6924.2009.01125.x). URL: <https://doi.org/10.1111/j.1745-6924.2009.01125.x> (visited on 18/04/2023).
- [117] Nicholas Wade. 'Inquiry on Harvard Lab Threatens Ripple Effect'. en-US. In: *The New York Times* (Aug. 2010). ISSN: 0362-4331. URL: <https://www.nytimes.com/2010/08/13/education/13harvard.html> (visited on 18/04/2023).

- [118] Huibo Wang et al. ‘Running Language Interpreters Inside SGX: A Lightweight, Legacy-Compatible Script Code Hardening Approach’. In: *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*. Asia CCS ’19. New York, NY, USA: Association for Computing Machinery, July 2019, pp. 114–121. ISBN: 978-1-4503-6752-3. DOI: [10.1145/3321705.3329848](https://doi.org/10.1145/3321705.3329848). URL: <https://doi.org/10.1145/3321705.3329848> (visited on 07/10/2022).
- [119] Xiao Wang, Alex J. Malozemoff and Jonathan Katz. *Faster Secure Two-Party Computation in the Single-Execution Setting*. Report Number: 762. 2016. URL: <https://eprint.iacr.org/2016/762> (visited on 21/03/2023).
- [120] Xiao Wang, Samuel Ranellucci and Jonathan Katz. *Authenticated Garbling and Efficient Maliciously Secure Two-Party Computation*. Report Number: 030. 2017. URL: <https://eprint.iacr.org/2017/030> (visited on 21/03/2023).
- [121] Xiao Wang, Samuel Ranellucci and Jonathan Katz. *Global-Scale Secure Multiparty Computation*. Report Number: 189. 2017. URL: <https://eprint.iacr.org/2017/189> (visited on 21/03/2023).
- [122] Andrew C. Yao. ‘Protocols for secure computations’. In: *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*. ISSN: 0272-5428. Nov. 1982, pp. 160–164. DOI: [10.1109/SFCS.1982.38](https://doi.org/10.1109/SFCS.1982.38).
- [123] Samee Zahur and David Evans. *Obliv-C: A Language for Extensible Data-Oblivious Computation*. Report Number: 1153. 2015. URL: <https://eprint.iacr.org/2015/1153> (visited on 21/03/2023).
- [124] Nezer Zaidenberg et al. ‘Trusted Computing and DRM’. en. In: *Cyber Security: Analytics, Technology and Automation*. Ed. by Martti Lehto and Pekka Neittaanmäki. Intelligent Systems, Control and Automation: Science and Engineering. Cham: Springer International Publishing, 2015, pp. 205–212. ISBN: 978-3-319-18302-2. DOI: [10.1007/978-3-319-18302-2\\_13](https://doi.org/10.1007/978-3-319-18302-2_13). URL: [https://doi.org/10.1007/978-3-319-18302-2\\_13](https://doi.org/10.1007/978-3-319-18302-2_13) (visited on 01/03/2023).
- [125] Ning Zhang et al. ‘TruSense: Information Leakage from TrustZone’. In: *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*. Apr. 2018, pp. 1097–1105. DOI: [10.1109/INFOCOM.2018.8486293](https://doi.org/10.1109/INFOCOM.2018.8486293).
- [126] Wei Zheng et al. ‘A survey of Intel SGX and its applications’. en. In: *Frontiers of Computer Science* 15.3 (Dec. 2020), p. 153808. ISSN: 2095-2236. DOI: [10.1007/s11704-019-9096-y](https://doi.org/10.1007/s11704-019-9096-y). URL: <https://doi.org/10.1007/s11704-019-9096-y> (visited on 02/03/2023).