Institute of Information Security

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

MPC Protocols For Comparisons

Timm Marquardt

Course of Study:	Informatik
Examiner:	Prof. Dr. Ralf Küsters
Supervisor:	Marc Rivinius, M.Sc.
Commenced:	November 10, 2021
Completed:	May 10, 2022

Abstract

Multi-party computation allows to compute functions on private inputs without revealing any information about input or output. Secure comparisons are an important building block for many functions and find application in many MPC deployments like secret bidding at auctions, privacy-preserving machine learning, secure linear programming or secure data mining. In recent years, many different MPC protocols for comparisons have been proposed for various different MPC settings. We collect the most promising proposed n-party protocols, compatible with SPDZ or $SPDZ_{2^k}$ and evaluate these regarding different MPC settings and cost parameters. The MPC settings include the protocols security model, adversary model, majority model and arithmetic domain, in which computation takes place. The costs are evaluated by determining necessary rounds, communication, computation and precomputation. We provide an overview of the MPC settings and costs for each protocol. Additionally, we contribute implementations in MP-SPDZ and benchmark tests in multiple different MPC settings for most of the chosen protocols. Our results show the advantages and disadvantages of each protocol and enable quick decisions for which protocol to choose for a specific application.

Contents

List of Figures

List of Tables

1 Introduction

In 1982, Yao [31] formulated the problem of comparing two private values in a privacy-preserving manner as the millionaire's problem. In the millionaires' problem, two millionaires want to find out who is richer without revealing their actual wealth. Since the problems' introduction, many different solutions have been proposed for the 2-party, 3-party and n-party setting, with $n \in \mathbb{N}$.

Multi-party computation (MPC), sometimes called secure multi-party computation (SMC), can be used to solve the millionaires' problem in the n-party setting. Similar to Damgard et al. [7], we describe MPC as a technique to compute the secret output y_i of a function f with private inputs x_i from different parties $P_i, ..., P_n$ for $i \in \{1, ..., n\}$ and $y_i, ..., y_n = f(x_i, ..., x_n)$. The challenge of MPC lies in computing the function f efficiently, regarding communication and computation, without revealing any information about input or output. While research on MPC has been focused on solutions for efficient arithmetic operations like addition and multiplication, other important primitive operations, like comparisons, which can be used to solve the millionaire's problem, are still very expensive compared their insecure counterpart.

The range of deployment of privacy-preserving applications is increasing at a high pace and already reaches very distinct areas like secret bidding at auctions [4], privacy-preserving machine learning [21], secure linear programming [27] and secure data mining [3]. The importance of efficient secure comparisons rises steadily because these are the bottleneck in the named areas and the amount of available data for computation reaches new heights every day. In recent years, various research has been conducted to increase the efficiency of secure comparisons, however, the solutions are often limited to specific MPC settings, e.g. adversary model and bound to drawbacks like statistical security or a larger arithmetic domain for computations.

In this work, we collect the vast amount of MPC protocols for comparisons in the n-party setting compatible with SPDZ [9] or SPDZ_{2^k} [6], filter out the most promising ones, analyze them and provide an overview of their performances in different environments. First, we present related work for MPC and secure comparisons, Chapter 2. We follow with introducing several levels, on which we analyze the protocols, to point out the differences in their constructions and compare their theoretical costs based on other MPC primitives, Chapter 3. The different levels include the MPC settings (security model, adversary model, majority model and arithmetic domain) and protocol cost evaluation (rounds, communication, computation, precomputation). Next, we present our detailed analysis of the protocols construction and theoretical costs in our protocol collection, Chapter 4. If possible, we also provide implementations of the protocols in the MPC framework MP-SPDZ [11]. We benchmark the protocols in MP-SPDZ in multiple different MPC settings and provide a detailed analysis of their performances, Chapter 5. Further, we discuss the results of this work by evaluating the advantages and disadvantages of the protocols for different environments, Chapter 6 and finish our work with a conclusion, Chapter 7.

2 Related Work

Yao's [31] introduction of the millionaires' problem in 1982, in which two millionaires try to find out who is richer, without revealing their actual wealth, has become a cornerstone for secure comparisons. While the millionaires' problem only considers two parties who know their own wealth, secure comparisons exist for the extension of the problem to the n-party setting with the actual wealth being unknown to anyone.

Various research on secure comparison protocols for the specific 2-party or 3-party setting has been conducted by [19], [29], [30] and [14]. Additionally, Shi et al. [26] present a quantum cryptographic comparison protocol for the 2-party setting. We do not cover any of their research in our work, because we focus on secure comparison protocols for the more general n-party setting.

Damgard et al. [8] proposed the first unconditionally secure constant round MPC protocol for comparisons in the n-party setting. While their protocol was still lacking in efficiency, parts of their work has been widely used and improved in the following years by [22], [24], [23], [5], [28], [18], [15], [7], [13], [12] and [20]. We collect the most promising proposed MPC protocols for comparisons of previously mentioned work and analyze these, to compare them with each other in different MPC settings and environments.

With more and more frameworks for MPC coming into realization in recent years, the MPC protocol evaluation has moved from theoretical cost analysis to benchmarks. In this work, we provide both - theoretical cost analysis and benchmarks. Hastings et al. [16] evaluated eleven different MPC frameworks, which have been released previous to the second half of 2018. For our work, we have decided to utilize the framework MP-SPDZ published by Data61 in 2019 [11]. Keller [17] provides further information on the MPC framework. MP-SPDZ was built to benchmark MPC protocols in many different MPC settings and thus perfectly suits the purpose of this work. We implement the analyzed protocols in MP-SPDZ, if possible.

The general field of application requiring MPC includes a wide range of different environments, however, MPC protocols for comparisons are especially important for areas like secret bidding at auctions [4], privacy-preserving machine learning [21], secure linear programming [27] or secure data mining [3]. Optimization of MPC protocols for comparisons has a direct impact on the efficiency in these areas and can increase the throughput of large data sets significantly.

3 Preliminaries

The related work on MPC protocols for comparisons is too vast to consider it all. This chapter first introduces general aspects of MPC protocols, as well as important notations used in this work, Section 3.1. Following this, the different MPC settings for MPC protocols are given, Section 3.2. Finally, the metrics used to evaluate the different costs of MPC protocols for comparisons are presented, Section 3.3. We adopt the notations used by relevant scientific literature related to our work.

3.1 Secret Sharing for MPC

MPC is usually based on secret sharing, e.g. Shamir's secret sharing [25], homomorphic encryption, e.g. somewhat homomorphic encryption [10] or Yao's garbled circuits. In this work, comparison protocols refer to MPC protocols for comparisons. In our case, we focus on MPC protocols based on secret sharing schemes in the n-party setting. We note, that some of the introduced comparison protocols can also be applied to MPC based on homomorphic encryption.

$\mathbb{Z}_M$ is the arithmetic domain, in which all computations in the arithmetic circuit take place. It is possible to secretly *share* a value $a \in \mathbb{Z}_M$ across all parties. We denote a secret shared value $a \in \mathbb{Z}_M$ as $[a] \in \mathbb{Z}_M$. Each party P_i for $i \in \{1, ..., n\}$ then holds a share a_i of $[a]$. To *reveal* $[a]$, the parties need to combine their shares a_i by communicating with each other. The number of parties required to reveal a secret shared value depends on the underlying secret sharing scheme. Secret sharing schemes allow for the three arithmetic operations $[a] + b$, $[a] + [b]$ and $[a] \cdot b$ to be calculated locally. In this work, we refer to these three as *arithmetic operations*. However, the multiplication of two secret shared values $[a] \cdot [b]$ requires communication between the parties. In this work, *multiplication* always refers to multiplication of two secret shared values. MPC protocols may provide different implementations to compute $[a] \cdot [b]$.

Some protocols make use of a binary circuit for the domain $\mathbb{Z}_2$ for efficient, non-linear operations like "xor", "or" and "negation". These operations are in general more expensive in the arithmetic circuit. To distinguish values in the binary circuit from values in the arithmetic circuit, we denote secret sharings in the binary circuit as $[a]_2 \in \mathbb{Z}_2$. Non-linear operations between two secret shared values in the binary circuit require communication between the parties. However, non-linear operations between a public value and a secret shared value can be performed locally. This is similar to multiplication in the arithmetic circuit.

We denote the bit-decomposition of a value $a \in \mathbb{Z}_M$ as $a_0, ..., a_{k-1}$ for $k \in \mathbb{N}$ and $a = \sum_{i=0}^{k-1} a_i \cdot 2^i$. Furthermore, a *bitwise shared value* consists of k secret sharings $[a_0], ..., [a_{k-1}] \in \{0, 1\}$ with $[a] = \sum_{i=0}^{k-1} [a_i] \cdot 2^i$ and $k \in \mathbb{N}$.

Based on multiplication, share, reveal and arithmetic operations, MPC usually provides protocols for the generation of different random components. Comparison protocols make direct use of these random components. Random components are secret shared when generated. We consider the following random components in our work: random bits, random bitwise shared values, random elements, random inverses and edaBits.

A random bit consists of the sharings of a bit $[a] \in \{0, 1\}$. Random bits can be used to generate random bitwise shared values, consisting of k random bits $[r_0], ..., [r_{k-1}]$ with $k \in \mathbb{N}$ and $[r] = \sum_{i=0}^{k-1} [r_i] \cdot 2^i$. Further, random elements are elements of the arithmetic domain $[r] \in \mathbb{Z}_M$. In some arithmetic domains, like prime fields, it is possible to generate inverses of elements. In these domains, it is possible to create a random element and its inverse $[r], [r]^{-1} \in \mathbb{Z}_M$.

Opposed to random bitwise shared values, Escudero et al. [13] introduce edaBits. EdaBits have a small, but important, difference compared to random bitwise shared values. An edaBit consists of a secret shared value in the arithmetic circuit $[a] \in \mathbb{Z}_M$ and k secret shared bits in the binary circuit $[a_0]_2, ..., [a_{k-1}]_2 \in \mathbb{Z}_2$, such that $a = \sum_0^{k-1} a_i \cdot 2^i$. EdaBits are usually created in large batches for efficient generation.

3.2 MPC Settings

The settings for MPC are usually separated by security model, adversary model, majority model and arithmetic domain, in which computation takes place. In general, all comparison protocols must be compatible with the models and domain of the underlying MPC protocols.

The security model can either be perfect security or statistical security. For perfect security, an adversary can not learn any secrets. For statistical security, an adversary can not learn any secret under some security parameter s. A default value for the security parameter s is 40. The parameter s is also often used as a correctness parameter.

The adversary model describes whether an adversary is either passive (semi-honest) or active (malicious). A passive adversary tries to obtain knowledge about secrets without deviating from any protocols. However, after protocol execution, a passive adversary can work together with other adversaries to try to reveal (parts of) secrets. An active adversary additionally might deviate from protocols. If an active adversary does deviate from any protocol, the execution aborts, which is also known as an active adversary with abort.

The majority model can either be an honest majority or a dishonest majority. The majority refers to the number of adversaries. For n parties and $2t < n$, an honest majority consists of at most t adversaries, while a dishonest majority consists of up to $n - 1$ adversaries. The security model is guaranteed under both majority models. Although some related works also distinguish between static and dynamic adversaries, we only consider the more standard, static adversaries. This means, that the adversaries do not change from one party to another during execution.

All computations take place in the arithmetic domain $\mathbb{Z}_M$. We generally distinguish between the arithmetic ring $\mathbb{Z}_M$ with $M = 2^k$ and $k \in \mathbb{N}$ and the arithmetic field $\mathbb{Z}_M$ with $M = p$ for a large odd prime p, sometimes also referred to as $\mathbb{F}_p$. The comparison protocols analyzed in this work are accurate for integers in $[0, 2^l - 1]$, $l \in \mathbb{N}$ and $2^{l+s+1} < M$. This allows for the reasonable condition, that all inputs, outputs and intermediate values are less than the chosen value for M.

3.3 Cost Evaluation of MPC Protocols

A standard approach to evaluate MPC protocol costs is the separation between an offline phase and an online phase. In the offline phase, precomputations are performed. This includes, but is not limited to, the generation of random components. These precomputations are independent of the data, on which the protocols operate. For the online phase, the protocols do require the data, on which they operate. In this work, we focus on the online phase, which varies a lot for comparison protocols. We note, that the offline phase is also relevant in practice, because the generation of random components can be very expensive depending on the MPC settings.

To analyze the different comparison protocols, we separate between five different metrics for cost estimation: online rounds, online communication, online computation, offline computation and random components. The cost often depends on a parameter k, which is the bit length of values in the arithmetic domain. Similar metrics are used by related works, however, most provide only the complexity but omit coefficients. Since the order of the complexity of most metrics is similar across many comparison protocols, we provide the costs as accurate as possible instead. Even though some older works have made similar approaches, many recent works focus on benchmarks instead of providing accurate, theoretical protocol costs.

The number of online rounds can dominate the execution time, especially in wide area networks (WAN). We consider one online round as one send and receive cycle in the online phase, in which parties can send an arbitrary amount of data to each other exactly once. Before, between and after the send and receive, parties can do any amount of local computation. This definition is widely used in related works and has been established by Ben-Or et al. [2]. In this work, we refer to online rounds as *rounds*.

The amount of online communication can be critical for networks with low bandwidth. Online communication between parties is often only measured by the number of multiplications in the online phase. We additionally include the number of reveals, because they have a similar online communication need as multiplications. The total online communication usually grows linearly with the number of parties.

Most related works omit the online computation, because it is either dominated by the online communication or by the number of online rounds. However, in a local area network (LAN) with high bandwidth and low round trip time, the online computation is an important factor. We measure the amount of online computation with the number of local arithmetic operations in the online phase. For example, addition of two secret shared values is considered as one arithmetic operation, because parties can compute it locally. We also assume, that computing the bit representation of a public value has the same cost as one arithmetic operation. We note, that arithmetic operations are often more expensive for protocols with the active adversary or dishonest majority model. The number of arithmetic operations required by multiplications and reveals depends on the underlying MPC protocols and is therefor not included in our calculation of online computation and has to be added manually.

Many comparison protocols require different random components. They include random bitwise shared values, random elements, random inverses, random bits and edaBits. These components can be generated in the offline phase. The generation of random components depends on the underlying MPC protocols and can vary a lot. We note, that the cost of generating random components is often

multiple times more expensive for protocols with the active adversary or dishonest majority model. This is often due to additional expensive techniques being added to the underlying secret sharing scheme.

The offline computation is calculated similar to the online computation. We measure the amount of offline computation with the number of local arithmetic operations in the offline phase. We additionally provide the number of necessary multiplications and reveals. Calculating constants for certain protocols has no cost, as these only have to be computed once and can be used for every protocol iteration afterwards. The number of arithmetic operations required by the generation of random components depends on the underlying MPC protocols and is therefor not included in our calculation of offline computation and has to be added manually.

4 Protocol Collection

This chapter contains the collection of the comparison protocols deeper analysed in this work. We first present some necessary and common sub-protocols for comparison protocols, Section 4.1. For the rest of this chapter, we present the most promising comparison protocols of related works for various MPC settings, Section 4.2 - 4.8, with some remarks at the end, Section 4.9. These comparison protocols can be deployed in different environments to suit security and cost characteristics of a diverse number of applications.

In case a related work introduces multiple similar comparison protocols, we always chose the protocol with the lowest (constant if possible) number of rounds. The other introduced versions usually trade a few rounds for communication. However, because the complexity order of rounds and communication stays the same, we do not further analyze those protocols in our work.

We put a focus on the MPC settings and cost evaluation introduced in Section 3.2 and 3.3 and point out the basic ideas behind the protocols. We note, that we conduct our analysis on the raw protocols, not the implementations. Due to the limitations of the framework MP-SPDZ [11], some of our implementations have different online and offline costs compared to the raw protocols.

For reference, on modern CPUs a common insecure comparison $a < b$ of two positive values a and b is determined in one clock cycle by the arithmetic logic unit. The arithmetic logic unit calculates $a - b$ and sets the negative flag, if an overflow occurs. The negative flag indicates whether $a < b$.

$$(a < b) = \begin{cases} 1, & ((a-b) < 0) = 1, \\ 0, & ((a-b) < 0) = 0 \end{cases}$$

4.1 Sub-Protocols

We present the most important sub-protocols for the comparison protocols in this section. While many protocols either use a common prefix-multiplication, Section 4.1.1 or a common bitwise-carry-add, Section 4.1.2, a new bitwise-less-than sub-protocol is introduced for almost every comparison protocol, Section 4.1.3.

4.1.1 Prefix-Multiplication

Catrina and de Hoogh [5] present a constant round prefix-multiplication protocol (PreMulC) based on the works of Bar-Ilan and Beaver [1]. For secret shared non-zero values $a_1, ..., a_k \in \mathbb{Z}_M$ with $M = p$ for a prime p and $j \in \{1, ..., k\}$, the prefix-multiplication protocol computes:

$$[q_j] = \prod_{i=1}^{j} [a_i]$$

The protocol has an online cost of 1 round, k reveals and $3k - 2$ arithmetic operations. Additionally, it requires $2k$ random elements, $2k - 1$ multiplications and $2k - 1$ arithmetic operations in the offline phase. We have used the implementation in MP-SPDZ [11] for our own implementations.

4.1.2 Bitwise-Carry-Add

The project SecureSCM [24] introduces a bitwise-carry-add, which calculates the bitwise shared sum and carry bit of two bitwise shared values $a, b \in \mathbb{Z}_M$ and $s = a + b$:

$$[s_1], ..., [s_{k+1}] \Leftarrow \text{bitwise-carry-add}([a_1], ..., [a_k], [b_1], ..., [b_k]).$$

As the prefix-multiplication protocol in Section 4.1.1 is not applicable to arithmetic rings $\mathbb{Z}_M$ with $M = 2^k$ for $k \in \mathbb{N}$, the bitwise-carry-add is often used as a replacement. The protocol has an online cost of $\log(k) + 1$ rounds, $k \log(k) + k$ multiplications and $7k - 1$ arithmetic operations. We have used the implementation in MP-SPDZ [11] for our own implementations.

4.1.3 Bitwise-Less-Than

The bitwise-less-than protocol compares a bitwise shared value $a \in \mathbb{Z}_M$ with a public value R:

$$([a] < R) = \begin{cases} 1, & \text{bitwise-less-than}([a_1], ..., [a_k], R) = 1, \\ 0, & \text{bitwise-less-than}([a_1], ..., [a_k], R) = 0 \end{cases}$$

This protocol is a very important building block to compare two secret shared values. It has been reconstructed, adapted and optimized to suit different MPC settings and cost characteristics. Each of the introduced protocols hereafter provides very different bitwise-less-than variations with different features and requirements. It is thus often not possible to use a specific bitwise-less-than protocol in a different MPC setting to the one introduced by the authors of the protocol. The bitwise-less-than protocol is often a critical part of comparison protocols.

4.2 Protocol by Reistad

Reistad [23] presents a comparison protocol against an active adversary in the dishonest majority setting. His protocol provides perfect security over the arithmetic field $\mathbb{Z}_M$ for positive integers in $[0, 2^l - 1]$, $l \in \mathbb{N}$ and $2^{l+s+1} < M = p$.

The protocol requires the prefix-multiplication sub-protocol. Together with the comparison protocol introduced by Reistad, this adds up to a total online cost of 5 rounds, 2 multiplications, $k + 2$ reveals and $12k + 38$ arithmetic operations and additionally 2 random bitwise shared values, $2k$ random elements, $2k - 1$ multiplications and $2k - 1$ arithmetic operations in the offline phase.

The basic idea of the protocol is to turn the comparison of two secret shared values $a, b \in \mathbb{Z}_M$ into extracting the least significant bit (LSB) of $z = 2 \cdot (a - b)$ and was first introduced by Nishide and Ohta [22]. The LSB determines whether $2 \cdot (a - b)$ is reduced by modulo p. This is only the case if $a - b$ is less than zero and thus the least significant bit is 1, as p is odd.

```
def LT(a, b):
    z = a - b                                                    #1 AOP

    r, r_bits = get_random_bitwise_shared_value(k)               #1 Random Bitwise Shared Value (Offline)

    c = (2 * z + r).reveal()                                     #2 AOP, 1 Reveal, 1 Round

    c_bits = c.bit_decompose(k)                                  #1 AOP

    x0 = GTBits(r_bits, c_bits)                                  #1 GTBits

    res = c_bits[0].bit_xor(r_bits[0][0]).bit_xor(x0)            #7 AOP, 1 MUL, 1 Round
    return res


def GTBits(r_bits, c_bits):
    tmp_vec = [sint(1)] * k
    for i in range(1, k):
        tmp_vec[i] = 1 + c_bits[k - i].bit_xor(r_bits[k - i][0])  #(5k - 5) AOP

    exponent = PreMulC(tmp_vec)                                  #PreMulC by Catrina and de Hoogh [5]

    x = sint(0)
    for i in range(k):                                           #4k AOP
        x += r_bits[k - 1 - i][0].bit_and(1-c_bits[k - 1 - i]) * exponent[i]

    return LSB(x)                                                #1 LSB


def LSB(x):
    s, s_bits = get_random_bitwise_shared_value(k)               #1 Random Bitwise Shared Value (Offline)

    d = (s + x).reveal()                                         #1 AOP, 1 Reveal, 1 Round
    d_bits = d.bit_decompose(k)                                  #1 AOP

    s12 = s_bits[k-1][0] * s_bits[k-2][0]                        #1 Mul, 1 Round

    d1_0 = (1 - s_bits[k-1][0] - s_bits[k-2][0] + s12) * d_bits[0]  #4 AOP
    + (s_bits[k-2][0] - s12) * (d_bits[0].bit_xor(d < 2**(k-2)))    #8 AOP
    + (s_bits[k-1][0] - s12) * (d_bits[0].bit_xor(d < 2**(k-1)))    #8 AOP
    + s12 * (d_bits[0].bit_xor(d < (2**(k-1)+2**(k-2))))            #8 AOP

    x0 = s_bits[0][0].bit_xor(d1_0)                              #4 AOP
    return x0
```

Figure 4.1: Implementation in MP-SPDZ of the Comparison Protocol by Reistad [23]

$$(a < b) = \begin{cases} 1, & \text{LSB}(2 \cdot (a - b)) = 1, \\ 0, & \text{LSB}(2 \cdot (a - b)) = 0 \end{cases}$$

To extract the LSB of z, a constant round bitwise-less-than sub-protocol, which uses the prefix-multiplication sub-protocol, is used. We provide an implementation in MP-SPDZ [11] with our cost calculation annotations in Figure 4.1.

4.3 Protocol by Catrina and de Hoogh

Catrina and de Hoogh [5] present a comparison protocol against a passive adversary in the honest majority setting. Their protocol provides statistical security over the arithmetic field $\mathbb{Z}_M$ for positive and negative integers in $[-2^l, 2^l - 1]$, $l \in \mathbb{N}$ and $2^{l+s+1} < M = p$. It can be extended for fixed-point rational numbers.

The protocol requires the prefix-multiplication sub-protocol. The combined online cost results in 3 rounds, $k + 2$ reveals and $10k + 17$ arithmetic operations and additionally 2 random bitwise shared values, $2k + 2$ random elements, $2k - 1$ multiplications and $2k - 1$ arithmetic operations in the offline phase.

The basic idea of the protocol is to determine whether the difference between two secret shared values $a, b \in \mathbb{Z}_M$ is less than zero. To calculate this, Catrina and de Hoogh introduce a truncation protocol to obtain the most significant bit (MSB) of $a - b$, since it indicates whether $a - b < 0$.

$$(a < b) = \begin{cases} 1, & \text{MSB}(a - b) = 1, \\ 0, & \text{MSB}(a - b) = 0 \end{cases}$$

The truncation protocol is based on a constant round bitwise-less-than sub-protocol, which uses the prefix-multiplication sub-protocol. As this comparison protocol is already implemented in MP-SPDZ [11], we do not provide our own implementation. However, we have conducted a similar analysis compared to the other protocols to calculate the online and offline costs.

4.4 Protocol by Lipmaa and Toft

Lipmaa and Toft [18] present a comparison protocol against an active adversary in the honest majority setting. Their protocol provides perfect security over the arithmetic field $\mathbb{Z}_M$ for positive and negative integers in $[-2^l, 2^l - 1]$, $l \in \mathbb{N}$ and $2^{l+s+1} < M = p$.

The protocol requires the prefix-multiplication sub-protocol. However, the prefix-multiplication sub-protocol is only used in the offline phase. The combined online cost results in approximately $5\log(k)$ rounds, $2\log(k)$ multiplications, $3\log(k)$ reveals and $16k + 30\log(k)$ arithmetic operations and additionally $3\log(k)$ random bitwise shared values, $4k + 4\log(k)$ random elements, $\log(k)$ random inverses, $2k$ random bits, $4k - \log(k)$ multiplications, $2k + 2\log(k)$ reveals and $24k + 16\log(k)$ arithmetic operations in the offline phase. We note that even though the number of required random objects is estimated accurately, the random objects may differ in their length k. Further, we note that both online and offline costs might appear more expensive at first, however, the online communication cost is logarithmic.

The basic idea of the protocol is to recursively check whether the $\frac{k}{2}$ upper bits of two secret shared values $a, b \in \mathbb{Z}_M$, both of length k, are different, until only the highest bit, in which a and b differ, is left. The remaining bit indicates whether $a < b$. To check whether the $\frac{k}{2}$ upper bits of a and b are different, an equal-zero sub-protocol, based on the hamming distance between a and b, is introduced. We provide an implementation in MP-SPDZ [11] with our cost calculation annotations in Figure 4.2. For simplicity, our protocol is implemented with bit length $k = 2^n$ with $n \in \mathbb{N}$ and a Mersenne prime p.

```
def eqH(x, k):
    r, r_bits = get_random_bitwise_shared_value(k)                    # 1 Random Bitwise Shared Value (Offline)
    R, R1 = sint.get_random_inverse()                                 # 1 Random Inverse (Offline)

    R_vec = [R] * k
    R_vec = PreMulC(R_vec)                                            # PreMulC by Catrina and de Hoogh [5] (Offline)

    m = (r + x).reveal()                                              # 1 AOP, 1 Reveal, 1 Round
    m = m.bit_decompose(k)                                            # 1 AOP

    H_1 = [sint(1)] * (k+1)
    for i in range(k):                                                # 5k AOP
        H_1[0] += m[i] + r_bits[i][0] - 2 * m[i] * r_bits[i][0]
    mH = (R1 * H_1[0]).reveal()                                       # 1 MUL, 1 Reveal, 2 Rounds

    H_1[0] = sint(1)
    for i in range(1, k+1):                                           # 2k AOP
        H_1[i] = mH**i * R_vec[i - 1]

    res = sint(0)
    for i in range(k+1):                                              # 2(k+1) AOP
        res += alpha[i] * H_1[i]                                      # alpha[i] precomputed constant

    return res


def GTonline(x, y, k):
    if (k == 1):                                                      # 1 AOP
        return 1 - y + x * y

    r = [sint(0)] * k
    for i in range(k):                                                # k Random Bit (Offline)
        r[0] = sint.get_random_bit()

    r_lower = sint(0)
    r_upper = sint(0)
    for i in range(int(k/2)):                                         # 3.5k AOP
        r_lower += 2**i * r[i]
        r_upper += 2**i * r[i + int(k/2)]

    r_A_k = sint.get_random_int(k)                                    # 1 Random Element (Offline)
    r_B_k = sint.get_random_int(k)                                    # 1 Random Element (Offline)

    Rk = 2**k * (r_A_k + r_B_k) + 2**int(k/2) * r_upper + r_lower     # 5 AOP

    z = 2**k + x - y                                                  # 2 AOP
    m = (z + Rk).reveal()                                             # 1 AOP, 1 Reveal, 1 Round
    m_lower = m % (2**int(k/2))                                       # 1 AOP
    m_upper = (m / (2**int(k/2))) % (2**int(k/2))                     # 2 AOP

    b = eqH(m_upper - r_upper, int(k/2))                              # 1 eqH
    m1 = b * (m_lower - m_upper) + m_upper                            # 3 AOP
    r1 = b * (r_lower - r_upper) + r_upper                            # 2 AOP, 1 MUL, 1 Round
    f = 1 - (GTonline(m1, r1, int(k/2)))                              # 1 AOP, GTonline with k = k/2
                                                                      # 6 AOP
    zmod2k = (m % (2**k) - (2**(int(k/2)) * r_upper + r_lower)) + 2**k * f

    return pow(2**k, -1, prime) * (z - zmod2k)                        # 2 AOP
```

Figure 4.2: Implementation in MP-SPDZ of the Comparison Protocol by Lipmaa and Toft [18]

4.5 Protocol by Goss and Jiang

Goss and Jiang [15] present an asymmetric comparison protocol against a passive adversary with two mutually incorruptible parties and additive secret sharing scheme. Their protocol provides perfect security over the arithmetic ring $\mathbb{Z}_M$ for positive integers in $[0, 2^l - 1]$, $l \in \mathbb{N}$ and $2^{l+s+1} < M = 2^k$.

Despite the protocol not quite fitting into our usual MPC setting by requiring two mutually incorruptible parties and additive secret sharing, instead of an honest- or dishonest majority and any secret sharing scheme, we have decided to analyze it in this work, as the asymmetric nature allows for new possibilities in MPC. Additionally, we add a bit-decomposition protocol because the protocol by Goss and Jiang operates on shared and bit-decomposed inputs. The bit-decomposition protocol, however, limits the otherwise required arithmetic group to an arithmetic ring.

For our analysis, we have used the bit-decomposition protocol introduced by Damgard et al. [7], which requires the bitwise-carry-add protocol. Additionally, we need to adapt our cost analysis to the asymmetric protocol. We consider sending a value to a different party as the same cost as revealing a value. We note, however, that no additional arithmetic operations are required after sending a value, which is implied by the reveal operation of the other protocols. The combined online cost results in $\log(k) + 7$ rounds, $k \log(k)$ multiplications, $12k + 8$ reveals and $47.5k + 13$ arithmetic operations and additionally $2k - 1$ random elements, $4k + 1$ random bits and $2k$ arithmetic operations in the offline phase.

The basic idea of the protocol is that two mutually incorruptible parties hold all the shares of two secret shared values $a, b \in \mathbb{Z}_M$. Instead of comparing their values directly with one another, they use a third party for the comparison. With the use of random values, they mask a and b in such a way, that the third party can compute a result of $a < b$ with neither knowing the values a or b, nor knowing the result of the comparison. The third party simply outputs another masked value, which is turned into a secret share of the result in return. The importance of the requirement of two mutually incorruptible parties in this comparison protocol can not be understated, because these two parties could reveal the secret values by themselves, if they were colluding. We do not provide an implementation for this protocol in MP-SPDZ, because we can not find a way to implement an asymmetric protocol in MP-SPDZ.

4.6 Protocol by Damgard et al.

Damgard et al. [7] present a comparison protocol against an active adversary in the dishonest majority setting. Their protocol provides statistical security over the arithmetic ring $\mathbb{Z}_M$ for positive and negative integers in $[-2^l, 2^l - 1]$, $l \in \mathbb{N}$ and $2^{l+s+1} < M = 2^k$.

The protocol requires the bitwise-carry-add sub-protocol. The combined online cost results in $\log(k) + 4$ rounds, $k \log(k) + k$ multiplications, 3 reveals and $4k + 23$ arithmetic operations and additionally 1 random bitwise shared value, 2 random bits and 2 arithmetic operations in the offline phase. We note, that it is possible to use 1 edaBit instead of 1 random bitwise shared value, to avoid k conversions from the arithmetic domain $\mathbb{Z}_M$ to the binary domain $\mathbb{Z}_2$. We have used 1 edaBit for our implementation.

The basic idea of the protocol is identical to the one introduced by Catrina and de Hoogh [5] and it is to check whether the difference between two secret shared positive values $a, b \in \mathbb{Z}_M$ is less than zero. However, instead of using a truncation protocol, Damgard et al. directly extract the most significant bit (MSB) of $a - b$, which indicates whether $a - b < 0$.

$$(a < b) = \begin{cases} 1, & \text{MSB}(a-b) = 1, \\ 0, & \text{MSB}(a-b) = 0 \end{cases}$$

The extraction of the MSB is based on a logarithmic round bitwise-less-than sub-protocol, which requires the bitwise-carry-add sub-protocol. We provide an implementation in MP-SPDZ [11] with our cost calculation annotations in Figure 4.3.

4.7 Protocol by Duan et al.

Duan et al. [12] present a comparison protocol against an active adversary in the honest majority setting. Their protocol provides perfect security over the arithmetic field $\mathbb{Z}_M$ for positive and negative integers in $[-2^l, 2^l - 1]$, $l \in \mathbb{N}$ and $2^{l+s+1} < M = p$. It can be extended for fixed-point rational numbers.

The online cost for their comparison protocol is $\log(k) + 2$ rounds, $2k + 2$ multiplications, 1 reveal and $8k + 13$ arithmetic operations and additionally 1 random bitwise shared value in the offline phase.

The basic idea of the protocol is identical to the one used by Reistad [23], first introduced by Nishide and Ohta [22] and it is to turn the comparison of two secret shared values $a, b \in \mathbb{Z}_M$ into extracting the least significant bit (LSB) of $z = 2 \cdot (a - b)$. The LSB determines whether $2 \cdot (a - b)$ is reduced by modulo p. This is only the case if $a - b$ is less than zero and thus the least significant bit is 1, as p is odd.

$$(a < b) = \begin{cases} 1, & \text{LSB}(2 \cdot (a-b)) = 1, \\ 0, & \text{LSB}(2 \cdot (a-b)) = 0 \end{cases}$$

Opposed to the bitwise-less-than sub-protocol by Reistad [23], Duan et al. use a different bitwise-less-than sub-protocol to extract the LSB of z, which does not require the prefix-multiplication sub-protocol. Their bitwise-less-than sub-protocol, however, needs $\log(k)$ rounds. We provide an implementation in MP-SPDZ [11] with our cost calculation annotations in Figure 4.4. For simplicity, our protocol is implemented with bit length $k = 2^n$ with $n \in \mathbb{N}$ and a Mersenne prime p.

```
def LT(a, b):
    return MSB(a-b)                                          # 1 AOP, 1 MSB


def MSB(a):
    r, r_bits = sint.get_edabit(k, True)                     # 1 EdaBit (Offline)
    b = sint.get_random_bit()                                # 1 Random Bit (Offline)

    r1 = r - sint(r_bits[k-1]) * 2**(k-1)                    # 1 AOP
    c = ((a+r)).reveal()                                     # 1 AOP, 1 Reveal, 1 Round

    c1 = c % (2**(k-1))                                      # 1 AOP

    c1_bits = [cbit(0)] * (k-1)
    c1_tmp = c1
    for i in range(k-1):                                     # 1 AOP
        c1_bits[i] = cbit(c1_tmp % 2)
        c1_tmp >>= 1

    r_bits.pop(k-1)

    u2 = BitLT(c1_bits, r_bits)                              # 1 BitLT

    u = B2A(u2)                                              # 1 B2A

    a1 = c1 - r1 + 2**(k-1) * u                              # 3 AOP
    d = a - a1                                               # 1 AOP

    e = (d + 2**(k-1) * b).reveal()                          # 1 AOP, 1 Reveal, 1 Round, (1 AOP Offline)
    ek_1 = e >> k-1

    return (1 - (ek_1 + b - 2 * ek_1 * b))                   # 5 AOP


def A2B(x):
    return sbit(x % 2)                                       # 1 AOP


def B2A(x_bit):
    r = sint.get_random_bit()                                # 1 Random Bit (Offline)
    r2 = A2B(r)                                              # 1 A2B (Offline)
    c = cint((x_bit + r2).reveal())                          # 1 AOP, 1 Reveal, 1 Round
    return sint(c + r - 2 * c * r)                           # 4 AOP


def BitLT(a_bits, b_bits):
    b_bits2 = [sbit(0)] * (k-1)
    for i in range(k-1):                                     # k AOP
        b_bits2[i] = 1 - b_bits[i]

    return (1 - bitwise_carry_add(a_bits, b_bits2, 0, 1, 1)[k-1])          # 1 AOP, Bit-Adder by SecureSCM [24]
```

Figure 4.3: Implementation in MP-SPDZ of the Comparison Protocol by Damgard et al. [7]

4.8 Protocol by Makri et al.

Makri et al. [20] present a comparison protocol against an active adversary in the dishonest majority setting. Their protocol provides perfect security over the arithmetic ring $\mathbb{Z}_M$ for positive integers in $[0, 2^l - 1]$, $l \in \mathbb{N}$ and $2^{l+1} < M = 2^k$. It additionally provides statistical security over the arithmetic field $\mathbb{Z}_M$ for positive integers in $[0, 2^l - 1]$, $l \in \mathbb{N}$ and $2^{l+s+1} < M = p$. The different security model properties exist due to the nature of the edaBit generation. Note, that for the arithmetic ring, we have $2^{l+1} < M = 2^k$ instead of $2^{l+s+1} < M = 2^k$ compared to the other protocols. This is an important difference, which is only possible, because the security parameter s is not necessary for computation over the arithmetic ring. The security parameter s is not needed in this case, because the comparison protocol by Makri et al. [20] considers all overflows in $\mathbb{Z}_{2^k}$.

```
def LT(x, y):
    z = 2 * (x - y)                                                    # 2 AOP

    r, r_bits = get_random_bitwise_shared_value(k)                     # 1 Random Bitwise Shared Value (Offline)
    c = (z + r).reveal()                                               # 1 AOP, 1 Reveal, 1 Round
    c_bits = c.bit_decompose()                                         # 1 AOP

    w = BitComp(c_bits, r_bits, k)                                     # 1 BitComp

    cr1 = c_bits[0].bit_xor(r_bits[0][0])                              # 4 AOP

    b = w * (1 - cr1) + (1 - w) * (cr1)                                # 3 AOP, 2 MUL, 1 Round

    return b


def BitComp(r_bits, x_bits, k1):

    c = [sint(0)] * k1
    b = [sint(0)] * k1
    for i in range(k1):                                                # 7k AOP
        c[i] = x_bits[i][0] * (1 - r_bits[i])
        b[i] = 1 - r_bits[i] - x_bits[i][0] + 2 * r_bits[i] * x_bits[i][0]

    while(k1 > 1):                                                     # log(k) Rounds
        bc = [sint(0)] * int(k1/2)
        bb = [sint(0)] * int(k1/2)
        for i in range(1, int(k1/2)+1):                                # k MUL
            bc[i-1] = b[2*i-1] * c[2*i-2]
            bb[i-1] = b[2*i-1] * b[2*i-2]

        for i in range(1, int(k1/2)+1):
            c[i-1] = c[2*i-1] + bc[i-1]                                # k AOP
            b[i-1] = bb[i-1]

        k1 = int(k1/2)

    return c[0]
```

Figure 4.4: Implementation in MP-SPDZ of the Comparison Protocol by Duan et al. [12]

The protocol requires the bitwise-carry-add sub-protocol in the offline phase. Additionally, it requires a prefix-or sub-protocol. We have used the prefix-or by SecureSCM [24]. The combined online cost results in $\log(k)+1$ rounds, $3k\log(k)$ multiplications, 2 reveals and $15k+11$ arithmetic operations and additionally 2 edaBits, $k\log(k)+k$ multiplications and $7k-1$ arithmetic operations in the offline phase.

The basic idea of the protocol is that the sum of two secret shared values $a, b \in \mathbb{Z}_M$ modulo M is less than a and less than b, iff $a+b$ is reduced by M:

$$(a+b) \mod M = a+b-M\cdot \mathrm{LT}(a+b \mod M, a) = a+b-M\cdot \mathrm{LT}(a+b \mod M, b)$$

Using this characteristic, as well as the commutative property of addition in arithmetic fields, Makri et al. derive a formula to efficiently calculate $a < b$ with three parallel iterations of a logarithmic round bitwise-less-than sub-protocol. Their bitwise-less-than sub-protocol is based on the one introduced by Damgard et al. [8]. We provide an implementation for the domain $\mathbb{Z}_{2^k}$ in MP-SPDZ [11] with our cost calculation annotations in Figure 4.5.

```
def LTBits(R_int, x):
    R_tmp = R_int

    y = [sbit(0)] * len(x)
    z = [sbit(0)] * len(x)
    w = [sbit(0)] * len(x)
    c = sint(0)

    for i in range(len(x)):                                    # k AOP
        y[len(x) - 1 - i] = x[i].bit_xor(R_tmp % 2)
        R_tmp >>= 1

    R_tmp = R_int

    z = PreOpL(or_op, y)                                       # PreOr by SecureSCM [24]

    w[0] = z[0]
    for i in range(len(z) - 1):                                # k AOP
        w[len(z) - 1 - i] = z[len(z) - 1 - i] - (z[len(z) - 2 - i])

    for i in range(len(w)):                                    # 2k AOP
        c += sint(w[len(w) - 1 - i].bit_and(R_tmp % 2))
        R_tmp >>= 1

    return (1 - c)                                             # 1 AOP


def LTS(x, y):
    r, r_bits = sint.get_edabit(k, True)                       # 1 EdaBit (Offline)
    r1, r1_bits = sint.get_edabit(k, True)                     # 1 EdaBit (Offline)

    b = (y + r).reveal()                                       # 1 AOP, 1 Reveal, 1 Round
    a = (r1 - x).reveal()                                      # 1 AOP, 1 Reveal
    T = a + b                                                  # 1 AOP

    w1 = LTBits(b, r_bits)                                     # 1 LTBits
    w2 = LTBits(a, r1_bits)                                    # 1 LTBits
    w3 = (T < b)                                               # 1 AOP

    s = bitwise_carry_add(r_bits, r1_bits, 0, 0, 1)            # Bit-Adder by SecureSCM [24] (Offline)
    w4 = s.pop()
    w5 = LTBits(T, s)                                          # 1 LTBits

    return w1 + w2 + w3 - sint(w4) - w5                        # 4 AOP
```

Figure 4.5: Implementation in MP-SPDZ of the Comparison Protocol by Makri et al. [20]

4.9 Remarks

Our collection consists of the most promising MPC protocols for comparisons in the n-party setting, regarding the different MPC settings and cost evaluation introduced in Section 3.2 and 3.3. These protocols can be applied to a wide range of applications. We provide an overview of the different MPC settings for each protocol in Table 4.1 and an overview of the protocol costs in Table 4.2.

We note, that the comparison protocols in the domain $\mathbb{F}_p$, with a large odd prime p, are also applicable for the domain $\mathbb{Z}_q$ with q being the product of two large odd primes $q = p_1 \cdot p_2$. The in this way constructed q is also known as an RSA-modulus. The extension to the domain $\mathbb{Z}_q$ is possible, because the protocols usually require multiplicative inverses or a modulo reduction with an odd modulus. Both of the above mentioned arithmetic domains provide these features. However, some MPC settings, costs and implementations might change slightly.

Protocol	Adversary	Majority	Security	Domain
Reistad [23]	Active	Dishonest	Perfect	$\mathbb{F}_p$
Catrina and de Hoogh [5]	Passive	Honest	Statistical	$\mathbb{F}_p$
Lipmaa and Toft [18]	Active	Honest	Perfect	$\mathbb{F}_p$
Goss and Jiang [15]	Passive	2 mutually incorruptible parties*	Perfect	$\mathbb{Z}_{2^k}$
Damgard et al. [7]	Active	Dishonest	Statistical	$\mathbb{Z}_{2^k}$
Duan et al. [12]	Active	Honest	Perfect	$\mathbb{F}_p$
Makri et al. [20]	Active	Dishonest	Perfect**, Statistical	$\mathbb{Z}_{2^k}$, $\mathbb{F}_p$

Table 4.1: Overview of the Comparison Protocol Settings. *The comparison protocol by Goss and Jiang [15] requires two mutually incorruptible parties instead of an honest or dishonest majority. **Perfect security for the comparison protocol by Makri et al. [20] only holds for the domain $\mathbb{Z}_{2^k}$. In the domain $\mathbb{F}_p$, the protocol only provides statistical security.

Additionally we want to mention, that the protocols [15], [7] and [20] operate over binary and arithmetic circuits. The online communication appears a lot higher in those protocols, because we do not differentiate between the costs of a multiplication of two secret shared values in the binary circuit and the costs of a multiplication of two secret shared values in the arithmetic circuit. We note, that in those protocols, most multiplications refer to multiplications in the binary circuit.

Protocol	Online Rounds	Online Communication/ Computation	Offline Computation	Random Components
Reistad [23]	5	2 MUL, $k+2$ R/ $12k+38$ AOP	$2k-1$ MUL, $2k-1$ AOP	2 RBSV, $2k$ RE
Catrina and de Hoogh [5]	3	$k+2$ R/ $10k+17$ AOP	$2k-1$ MUL, $2k-1$ AOP	2 RBSV, $2k+2$ RE
Lipmaa and Toft [18]	$5\log(k)$	$2\log(k)$ MUL, $3\log(k)$ R/ $16k+30\log(k)$ AOP	$4k-\log(k)$ MUL, $2k+2\log(k)$ R, $24k+16\log(k)$ AOP	$3\log(k)$ RBSV, $4k+4\log(k)$ RE, $\log(k)$ RI, $2k$ RB
Goss and Jiang [15]	$\log(k)+7$	$k\log(k)$ MUL, $12k+8$ R*/ $47.5k+13$ AOP	$2k$ AOP	$2k-1$ RE, $4k+1$ RB
Damgard et al. [7]	$\log(k)+4$	$k\log(k)+k$ MUL, 3 R/ $4k+23$ AOP	2 AOP	1 RBSV (or 1 EdaBit), 2 RB
Duan et al. [12]	$\log(k)+2$	$2k+2$ MUL, 1 R/ $8k+13$ AOP	-	1 RBSV
Makri et al. [20]	$\log(k)+1$	$3k\log(k)$ MUL, 2 R/ $15k+11$ AOP	$k\log(k)+k$ MUL, $7k-1$ AOP	2 EdaBits

Table 4.2: Overview of the Comparison Protocol Costs. We have used the following abbreviations for the table entries: MUL stands for multiplication; R stands for reveal; AOP stands for arithmetic operations; RBSV stands for random bitwise shared value; RE stands for random Element; RI stands for random inverse; RB stands for random bit. *The reveal operation by Goss and Jiang [15] does not require any additional arithmetic operations in contrast to the reveal operation of the other protocols.

5 Benchmarks

We have run all our benchmarks on a single machine equipped with 16 GB RAM and an I7 processor with 6 cores. We use Windows Subsystem for Linux (WSL2) to run the MPC protocols on the Windows OS. In general, we use the 3-party setting for our benchmarks. We choose $k = 64$ for the arithmetic ring $\mathbb{Z}_{2^k}$ and a prime p with $\log(p) \approx 64$ for the arithmetic field $\mathbb{F}_p$. We additionally run our benchmarks for the protocol by Makri et al. [20] in the arithmetic ring $\mathbb{Z}_{2^k}$ with $k = 32$, because their comparison protocol enables a smaller arithmetic domain due to the possibility of eliminating the need for the security parameter s.

While protocols in the dishonest majority (active adversary) model are also applicable to the honest majority (passive adversary) model, this does not hold for the opposite. Therefore, we benchmark comparison protocols in their respective applicable MPC settings only. We measure the protocols performance in throughput (comparisons per second (ops/s)) and communication (kbits per comparison per party).

We note, that it is not possible to implement all MPC protocols for comparisons optimally in MP-SPDZ [11] due to the limitations of the framework. However, we have given our best efforts in the limited time of this work to design our implementations as efficient as possible without deviating too much from the original comparison protocols.

We benchmark the full comparison protocols in different MPC settings with $50,000$ comparisons per protocol per applicable MPC setting and present our results in Section 5.1. Further, we benchmark the online phase in the same setup, but with $500,000$ comparisons instead and show our results in Section 5.2. Lastly, we run the online phase of the comparison protocol by Damgard et al. [7] with different numbers of parties to show the influence of the number of parties on comparisons per second and communication per comparison. We provide our results thereof in Section 5.3.

5.1 Benchmarks of the Full Comparison Protocols

We present an overview of our benchmarks of the full comparison protocols (offline and online phase) in Table 5.1 and discuss the results in this section. We run $50,000$ comparisons in parallel per protocol per applicable MPC setting to achieve comparable results. We note, that we only run $5,000$ comparisons in parallel in the active adversary, dishonest majority setting for the comparison protocols by Damgard et al. [7] and Makri et al. [20], as the edaBit generation in the offline phase of these protocols requires more than 16 GB RAM for $50,000$ comparisons in MP-SPDZ. Our results are not optimal in this case due to the low number of only $5,000$ comparisons and the necessarily small bucket size [13] for the edaBit generation.

The throughput is up to multiple 100 times lower and the communication up to multiple 1,000 times higher in the active adversary, dishonest majority setting for all comparison protocols compared to the other MPC settings. The throughput only differs about 7 times and the communication only

Protocol	Measure	Dishonest Majority		Honest Majority	
		Active	Passive	Active	Passive
Reistad [23]	Thru. (ops/s)	39	233	2732	11904
	Comm. (kb)	5118.1	1104.2	31.2	4.4
Catrina and de Hoogh [5]	Thru. (ops/s)	-	-	-	1021
	Comm. (kb)	-	-	-	8.8
Lipmaa and Toft [18]	Thru. (ops/s)	-	-	533	2861
	Comm. (kb)	-	-	105.2	10.5
Damgard et al. [7]	Thru. (ops/s)	67	3674	16835	37878
	Comm. (kb)	2839.2	66.5	6.3	0.5
Duan et al. [12]	Thru. (ops/s)	-	-	5494	27521
	Comm. (kb)	-	-	13.1	1.8
Makri et al. [20] (64-bit)	Thru. (ops/s)	54	4108	9192	12260
	Comm. (kb)	3019.6	53.8	10.5	0.7
Makri et al. [20] (32-bit)	Thru. (ops/s)	99	6817	18293	23889
	Comm. (kb)	1584.8	30.8	4.6	0.4

Table 5.1: Benchmarks of the Full Comparison Protocols in Different MPC Settings

differs about 15 times between the other MPC settings. The throughput and communication of the passive adversary, dishonest majority setting in the arithmetic field $\mathbb{F}_p$ seem odd compared to our other results. We conclude, that this is most likely due to the underlying MPC protocols implemented in MP-SPDZ.

The overall best performances show the more recently proposed comparison protocols by Damgard et al. [7], Duan et al. [12] and Makri et al. [20]. Each of these protocols has their own favorable specific setting to be deployed in. The communication is mainly lower for comparison protocols over the arithmetic ring $\mathbb{Z}_{2^k}$ compared to protocols over the arithmetic field $\mathbb{F}_p$. In general, the comparison protocols by Catrina and de Hoogh [5] and Lipmaa and Toft [18] show very poor performances in both throughput and communication. Outside of the passive adversary, dishonest majority setting, the comparison protocol by Reistad [23] shows decent throughput and communication, but does not quite reach the performance of the more recently proposed comparison protocols. The comparison protocol by Makri et al. [20] is about twice as efficient for $k = 32$ compared to $k = 64$.

5.2 Benchmarks of the Online Phase

We present an overview of our benchmarks of the online phase of the comparison protocols in Table 5.2 and discuss the results in this section. We properly separate the online phase of each protocol using MP-SPDZ and run $500,000$ comparisons in parallel per protocol per applicable MPC setting to achieve comparable results.

The throughput is up to 7 times lower in the active adversary, dishonest majority setting for all comparison protocols, compared to the other MPC settings, where the throughput is within a factor of two. The communication is within a small margin for the active adversary, dishonest majority and the passive adversary, honest majority setting, as well as for the passive adversary, dishonest

Protocol	Measure	Dishonest Majority		Honest Majority	
		Active	Passive	Active	Passive
Reistad [23]	Thru. (ops/s)	5780	5513	6735	8704
	Comm. (kb)	4.5	4.5	8.1	1.9
Catrina and de Hoogh [5]	Thru. (ops/s)	-	-	-	-
	Comm. (kb)	-	-	-	-
Lipmaa and Toft [18]	Thru. (ops/s)	-	-	1620	1940
	Comm. (kb)	-	-	0.8	0.2
Damgard et al. [7]	Thru. (ops/s)	6558	38058	23890	41855
	Comm. (kb)	0.9	6.9	5.2	0.4
Duan et al. [12]	Thru. (ops/s)	-	-	15462	20912
	Comm. (kb)	-	-	4.7	0.9
Makri et al. [20] (64-bit)	Thru. (ops/s)	3485	13901	10243	13250
	Comm. (kb)	1.6	12.1	9.0	0.6
Makri et al. [20] (32-bit)	Thru. (ops/s)	7465	27568	20682	26662
	Comm. (kb)	0.7	4.9	3.7	0.3

Table 5.2: Benchmarks of the Online Phase of Comparison Protocols in Different MPC Settings

majority and the active adversary, honest majority setting. Between these two pairs of settings however, the communication differs up to 20 times. Unfortunately, it is not possible to separate the online and offline phases of the already in MP-SPDZ implemented comparison protocol by Catrina and de Hoogh [5]. Similar to the full comparison protocol benchmarks, the throughput and communication of the passive adversary, dishonest majority setting in the arithmetic field $\mathbb{F}_p$ seem odd compared to our other results. We conclude, that this is most likely due to the underlying MPC protocols implemented in MP-SPDZ.

The best throughput shows the comparison protocol by Damgard et al. [7], closely followed by the comparison protocol by Makri et al. [20]. The comparison protocol by Lipmaa and Toft [18] shows very poor throughput instead, but has the best communication of any protocol. The communication is in general slightly lower for comparison protocols over the arithmetic ring $\mathbb{Z}_{2^k}$ compared to protocols over the arithmetic field $\mathbb{F}_p$, outside of the comparison protocol by Lipmaa and Toft [18]. The comparison protocol by Duan et al. [12] has considerable throughput and communication, but does not outshine the other comparison protocols in any of these categories. While the comparison protocol by Reistad [23] is comparable in the active adversary, dishonest majority setting, it does lack in efficiency in the honest majority model. We note, that this might be due to the recursive character of the comparison protocol and an iterative implementation might improve the performance. It again holds true, that the comparison protocol by Makri et al. [20] is about twice as efficient for $k = 32$ compared to $k = 64$.

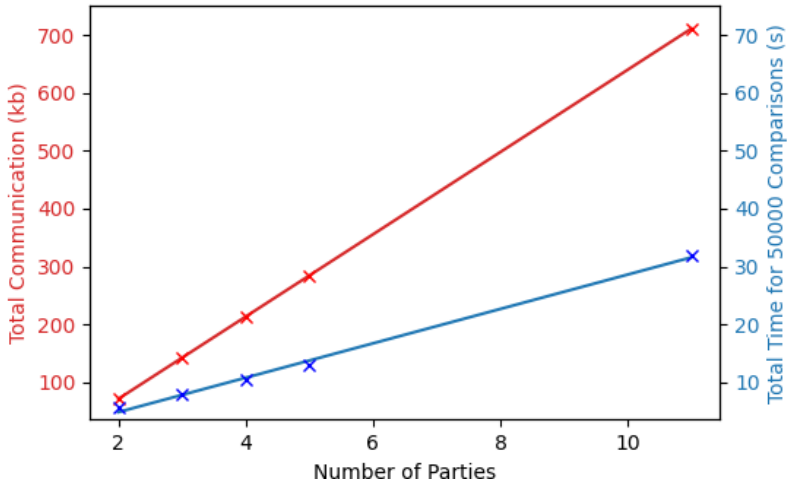

Figure 5.1: Total Time and Communication for Increasing Number of Parties

5.3 Influence of the Number of Parties

We present an overview of our benchmarks of the online phase of the comparison protocol by Damgard et al. [7] with different numbers of parties in Figure 5.1 and discuss the results in this section. Due to the nature of MPC protocols, the benchmark test would yield the same results for other protocols and settings. We run 50, 000 comparisons in parallel for 2, 3, 4, 5 and 11 parties.

Our results show a linear increase for total communication and total time for a linear increase in the number of parties. For our measurements this means, that for an increasing number of parties, the communication per party roughly stays the same, while the comparisons per second decrease. The communication per party is constant due to the nature of using star-shaped instead of direct communication. The comparisons per second decrease with an increasing number of parties, because we are running our benchmarks on a single machine. However, if we add a new machine for each new party, as it is often the case in practice, the comparisons per second would not decrease. We conclude, that the number of parties has no significant influence on our measurements of comparisons per second and communication per comparison per party.

6 Results

In this work, we analyze seven comparison protocols out of the vast research around MPC, benchmark six out of the seven comparison protocols and provide implementations for five in MP-SPDZ [11]. Overall, we focus on the online phase, but do not entirely disregard the offline phase. We present an overview of our theoretical cost analysis in Table 4.1 and Table 4.2 and an overview of our benchmarks in Table 5.1 and Table 5.2.

In the online phase, the throughput differs up to 7 times and the communication differs up to 20 times between the different MPC settings. For the offline phase however, the choice of the MPC setting has a much higher impact, as the throughput differs up to multiple 100 times and the communication up to multiple 1,000 times. The offline phase is especially expensive for the active adversary, dishonest majority setting. It is in general favorable to choose the least secure, but acceptable, MPC setting for the highest efficiency. We note, that we do not see any direct correlation between the throughput in our benchmarks and the online computation in our theoretical cost analysis. We are not exactly sure why this is, but it might be due to inefficient implementations on our side, or large overhead produced by the underlying MPC protocols implemented in MP-SPDZ.

The benchmarks of the comparison protocols by Damgard et al. [7] and Makri et al. [20] show, that the online communication of these protocols is indeed similar to other comparison protocols, despite the theoretical cost analysis indicating log-linear amount of communication, due to assuming multiplications in the binary circuit and arithmetic circuit as equivalent.

Utilizing the smaller size of values for the comparison protocol by Makri et al. [20], e.g. by setting $k = 32$ instead of $k = 64$, is not always feasible, due to other protocols potentially requiring a larger arithmetic domain $\mathbb{Z}_{2^k}$. However, in case a smaller arithmetic domain $\mathbb{Z}_{2^k}$ is possible, it should be used, as it increases the efficiency of all MPC protocols.

For the passive adversary, honest majority setting, the benchmarks of the full comparison protocols indicate slightly better efficiency over the separated online phase for some comparison protocols. This is most likely due to the separation process and underlying MPC protocols implemented in MP-SPDZ. We conclude, that the offline phase is very fast compared to the online phase in the passive adversary, honest majority setting.

The benchmarks demonstrate the overall best performances in throughput and communication for the two comparison protocols by Damgard et al. [7] and Makri et al. [20] ($k = 32$), which both operate over the arithmetic ring $\mathbb{Z}_{2^k}$. The comparison protocol by Duan et al. [12] follows closely behind, but operates over the arithmetic field $\mathbb{F}_p$. While most modern CPUs operate on 64-bit architectures and thus favor the use of the arithmetic ring $\mathbb{Z}_{2^k}$ in general, the arithmetic field $\mathbb{F}_p$ can not be disregarded, because it enables other primitives significant for specific applications, e.g. efficient division of secret shared values by public values. These three comparison protocols are the best choice for most environments.

The comparison protocol by Lipmaa and Toft [18] is the only comparison protocol with a logarithmic amount of communication in the online phase, which is also noticeable in the benchmarks. However, the offline phase is very expensive due to the high number of necessary random components. Further, the throughput is very low in our benchmarks. Thus, their protocol only makes a good choice, if communication in the online phase is the limiting factor by a large margin.

The comparison protocols by Reistad [23] and Catrina and de Hoogh [5] perform poorly in the benchmarks. However, referring to the theoretical cost analysis, these are the only constant round comparison protocols. Thus, if latency between parties is the limiting factor, these comparison protocols might demonstrate high efficiency in practice. We note, that the number of rounds are not represented in our benchmarks.

7 Conclusion and Outlook

In this work, we present the seven most promising MPC protocols for comparisons in the n-party setting, introduced by related works and compatible with SPDZ or $SPDZ_{2^k}$. We analyze the MPC protocols for comparisons and provide their theoretical costs and the MPC settings, in which they can be deployed in. Further, we provide implementations for five out of the seven and benchmark six out of the seven MPC protocols for comparisons.

Our benchmarks show, that the choice of the MPC setting has a much bigger influence on the offline phase compared to the online phase. Thus we conclude, that a more secure MPC setting should only be considered above the least secure, but acceptable, MPC setting, if it is possible to precompute the offline phase in a separate manner. In general however, the least secure, but acceptable, MPC setting achieves the highest efficiency for MPC protocols for comparisons. We demonstrate the best suited MPC protocols for comparisons for different MPC settings in our results, Chapter 6. Our decisions build upon the theoretical cost analysis and the benchmarks run for different MPC settings.

As we only focus on MPC protocols for comparisons in the n-party setting in this work, future research could include the various related work on MPC protocols for comparisons in the 2-party and 3-party setting and confront them with the most efficient protocols of this work. Further, the influence of the number of rounds on the throughput of comparisons in a wide area network (WAN) could be of interest and provides material for more benchmarks. Additionally, the different random components, sub-protocols and basic ideas, introduced by various research papers and presented in this work, may all serve as starting points to increase the efficiency of MPC protocols for comparisons in the future.

Bibliography

[1] J. Bar-Ilan, D. Beaver. "Non-Cryptographic Fault-Tolerant Computing in Constant Number of Rounds of Interaction". In: *Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing*. PODC '89. Edmonton, Alberta, Canada: Association for Computing Machinery, 1989, pp. 201–209. ISBN: 0897913264. DOI: 10.1145/72981.72995. URL: https://doi.org/10.1145/72981.72995 (cit. on p. 19).

[2] M. Ben-Or, S. Goldwasser, A. Wigderson. "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract)". In: *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*. Ed. by J. Simon. ACM, 1988, pp. 1–10. DOI: 10.1145/62212.62213. URL: https://doi.org/10.1145/62212.62213 (cit. on p. 17).

[3] D. Bogdanov, M. Niitsoo, T. Toft, J. Willemson. "High-performance secure multi-party computation for data mining applications". In: *Int. J. Inf. Sec.* 11.6 (2012), pp. 403–418. DOI: 10.1007/s10207-012-0177-2. URL: https://doi.org/10.1007/s10207-012-0177-2 (cit. on pp. 11, 13).

[4] P. Bogetoft, I. Damgård, T. P. Jakobsen, K. Nielsen, J. Pagter, T. Toft. "A Practical Implementation of Secure Auctions Based on Multiparty Integer Computation". In: *Financial Cryptography and Data Security, 10th International Conference, FC 2006, Anguilla, British West Indies, February 27-March 2, 2006, Revised Selected Papers*. Ed. by G. D. Crescenzo, A. D. Rubin. Vol. 4107. Lecture Notes in Computer Science. Springer, 2006, pp. 142–147. DOI: 10.1007/11889663_10. URL: https://doi.org/10.1007/11889663%5C_10 (cit. on pp. 11, 13).

[5] O. Catrina, S. de Hoogh. "Improved Primitives for Secure Multiparty Integer Computation". In: *Security and Cryptography for Networks, 7th International Conference, SCN 2010, Amalfi, Italy, September 13-15, 2010. Proceedings*. Ed. by J. A. Garay, R. D. Prisco. Vol. 6280. Lecture Notes in Computer Science. Springer, 2010, pp. 182–199. DOI: 10.1007/978-3-642-15317-4_13. URL: https://doi.org/10.1007/978-3-642-15317-4%5C_13 (cit. on pp. 13, 19, 21, 25, 29, 30, 32, 33, 36).

[6] R. Cramer, I. Damgård, D. Escudero, P. Scholl, C. Xing. "SPD$\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for Dishonest Majority". In: *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II*. Ed. by H. Shacham, A. Boldyreva. Vol. 10992. Lecture Notes in Computer Science. Springer, 2018, pp. 769–798. DOI: 10.1007/978-3-319-96881-0_26. URL: https://doi.org/10.1007/978-3-319-96881-0%5C_26 (cit. on p. 11).

[7] I. Damgård, D. Escudero, T. K. Frederiksen, M. Keller, P. Scholl, N. Volgushev. "New Primitives for Actively-Secure MPC over Rings with Applications to Private Machine Learning". In: *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*. IEEE, 2019, pp. 1102–1120. DOI: 10.1109/SP.2019.00078. URL: https://doi.org/10.1109/SP.2019.00078 (cit. on pp. 11, 13, 24, 26, 29–35).

[8] I. Damgård, M. Fitzi, E. Kiltz, J. B. Nielsen, T. Toft. "Unconditionally Secure Constant-Rounds Multi-party Computation for Equality, Comparison, Bits and Exponentiation". In: *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*. Ed. by S. Halevi, T. Rabin. Vol. 3876. Lecture Notes in Computer Science. Springer, 2006, pp. 285–304. DOI: 10.1007/11681878_15. URL: https://doi.org/10.1007/11681878%5C_15 (cit. on pp. 13, 27).

[9] I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, N. P. Smart. "Practical Covertly Secure MPC for Dishonest Majority - Or: Breaking the SPDZ Limits". In: *Computer Security - ESORICS 2013 - 18th European Symposium on Research in Computer Security, Egham, UK, September 9-13, 2013. Proceedings*. Ed. by J. Crampton, S. Jajodia, K. Mayes. Vol. 8134. Lecture Notes in Computer Science. Springer, 2013, pp. 1–18. DOI: 10.1007/978-3-642-40203-6_1. URL: https://doi.org/10.1007/978-3-642-40203-6%5C_1 (cit. on p. 11).

[10] I. Damgård, V. Pastro, N. P. Smart, S. Zakarias. "Multiparty Computation from Somewhat Homomorphic Encryption". In: *IACR Cryptol. ePrint Arch.* (2011), p. 535. URL: http://eprint.iacr.org/2011/535 (cit. on p. 15).

[11] Data61. *MP-SPDZ: Versatile Framework for Multi-party Computation*. [Online]. 2019. URL: https://github.com/data61/MP-SPDZ (visited on 04/08/2022) (cit. on pp. 11, 13, 19–22, 25, 27, 31, 35, 43).

[12] X. Duan, V. Goyal, H. Li, R. Ostrovsky, A. Polychroniadou, Y. Song. "ACCO: Algebraic Computation with Comparison". In: *CCSW@CCS '21: Proceedings of the 2021 on Cloud Computing Security Workshop, Virtual Event, Republic of Korea, 15 November 2021*. Ed. by Y. Zhang, M. van Dijk. ACM, 2021, pp. 21–38. DOI: 10.1145/3474123.3486757. URL: https://doi.org/10.1145/3474123.3486757 (cit. on pp. 13, 25, 27, 29, 30, 32, 33, 35).

[13] D. Escudero, S. Ghosh, M. Keller, R. Rachuri, P. Scholl. "Improved Primitives for MPC over Mixed Arithmetic-Binary Circuits". In: *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part II*. Ed. by D. Micciancio, T. Ristenpart. Vol. 12171. Lecture Notes in Computer Science. Springer, 2020, pp. 823–852. DOI: 10.1007/978-3-030-56880-1_29. URL: https://doi.org/10.1007/978-3-030-56880-1%5C_29 (cit. on pp. 13, 16, 31).

[14] W. Fujii, K. Iwamura, M. Inamura. "Secure Comparison and Interval Test Protocols based on Three-party MPC". In: *Proceedings of the 6th International Conference on Information Systems Security and Privacy, ICISSP 2020, Valletta, Malta, February 25-27, 2020*. Ed. by S. Furnell, P. Mori, E. R. Weippl, O. Camp. SCITEPRESS, 2020, pp. 698–704. DOI: 10.5220/0009161406980704. URL: https://doi.org/10.5220/0009161406980704 (cit. on p. 13).

[15] K. Goss, W. Jiang. "Efficient and Constant-Rounds Secure Comparison through Dynamic Groups and Asymmetric Computations". In: *IACR Cryptol. ePrint Arch.* (2018), p. 179. URL: http://eprint.iacr.org/2018/179 (cit. on pp. 13, 24, 29, 30).

[16] M. Hastings, B. Hemenway, D. Noble, S. Zdancewic. "SoK: General Purpose Compilers for Secure Multi-Party Computation". In: *2019 IEEE Symposium on Security and Privacy, SP 2019, San Francisco, CA, USA, May 19-23, 2019*. IEEE, 2019, pp. 1220–1237. DOI: 10.1109/SP.2019.00028. URL: https://doi.org/10.1109/SP.2019.00028 (cit. on p. 13).

[17] M. Keller. "MP-SPDZ: A Versatile Framework for Multi-Party Computation". In: *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*. Ed. by J. Ligatti, X. Ou, J. Katz, G. Vigna. ACM, 2020, pp. 1575–1590. DOI: 10.1145/3372297.3417872. URL: https://doi.org/10.1145/3372297.3417872 (cit. on p. 13).

[18] H. Lipmaa, T. Toft. "Secure Equality and Greater-Than Tests with Sublinear Online Complexity". In: *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part II*. Ed. by F. V. Fomin, R. Freivalds, M. Z. Kwiatkowska, D. Peleg. Vol. 7966. Lecture Notes in Computer Science. Springer, 2013, pp. 645–656. DOI: 10.1007/978-3-642-39212-2_56. URL: https://doi.org/10.1007/978-3-642-39212-2%5C_56 (cit. on pp. 13, 22, 23, 29, 30, 32, 33, 36).

[19] X. Liu, S. Li, J. Liu, X. Chen, G. Xu. "Secure multiparty computation of a comparison problem". In: *SpringerPlus* 5.1 (Sept. 2016). DOI: 10.1186/s40064-016-3061-0. URL: https://doi.org/10.1186/s40064-016-3061-0 (cit. on p. 13).

[20] E. Makri, D. Rotaru, F. Vercauteren, S. Wagh. "Rabbit: Efficient Comparison for Secure Multi-Party Computation". In: *Financial Cryptography and Data Security - 25th International Conference, FC 2021, Virtual Event, March 1-5, 2021, Revised Selected Papers, Part I*. Ed. by N. Borisov, C. Díaz. Vol. 12674. Lecture Notes in Computer Science. Springer, 2021, pp. 249–270. DOI: 10.1007/978-3-662-64322-8_12. URL: https://doi.org/10.1007/978-3-662-64322-8%5C_12 (cit. on pp. 13, 26, 28–33, 35).

[21] P. Mohassel, Y. Zhang. "SecureML: A System for Scalable Privacy-Preserving Machine Learning". In: *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE Computer Society, 2017, pp. 19–38. DOI: 10.1109/SP.2017.12. URL: https://doi.org/10.1109/SP.2017.12 (cit. on pp. 11, 13).

[22] T. Nishide, K. Ohta. "Multiparty Computation for Interval, Equality, and Comparison Without Bit-Decomposition Protocol". In: *Public Key Cryptography - PKC 2007, 10th International Conference on Practice and Theory in Public-Key Cryptography, Beijing, China, April 16-20, 2007, Proceedings*. Ed. by T. Okamoto, X. Wang. Vol. 4450. Lecture Notes in Computer Science. Springer, 2007, pp. 343–360. DOI: 10.1007/978-3-540-71677-8_23. URL: https://doi.org/10.1007/978-3-540-71677-8%5C_23 (cit. on pp. 13, 20, 25).

[23] T. I. Reistad. "Multiparty Comparison - An Improved Multiparty Protocol for Comparison of Secret-shared Values". In: *SECRYPT 2009, Proceedings of the International Conference on Security and Cryptography, Milan, Italy, July 7-10, 2009, SECRYPT is part of ICETE - The International Joint Conference on e-Business and Telecommunications*. Ed. by E. Fernández-Medina, M. Malek, J. Hernando. INSTICC Press, 2009, pp. 325–330 (cit. on pp. 13, 20, 21, 25, 29, 30, 32, 33, 36).

[24] SecureSCM. *Security Analysis*. EU FP7 Project Secure Supply Chain Management (SecureSCM). [Online]. 2009. URL: https://faui1-files.cs.fau.de/filepool/publications/octavian_securescm/SecureSCM-D.9.2.pdf (visited on 04/08/2022) (cit. on pp. 13, 20, 27).

[25] A. Shamir. “How to Share a Secret”. In: *Commun. ACM* 22.11 (1979), pp. 612–613. DOI: 10.1145/359168.359176. URL: http://doi.acm.org/10.1145/359168.359176 (cit. on p. 15).

[26] R. Shi, B. Liu, M. Zhang. “Secure two-party integer comparison protocol without any third party”. In: *Quantum Inf. Process.* 20.12 (2021), p. 402. DOI: 10.1007/s11128-021-03344-1. URL: https://doi.org/10.1007/s11128-021-03344-1 (cit. on p. 13).

[27] T. Toft. “Solving Linear Programs Using Multiparty Computation”. In: *Financial Cryptography and Data Security, 13th International Conference, FC 2009, Accra Beach, Barbados, February 23-26, 2009. Revised Selected Papers*. Ed. by R. Dingledine, P. Golle. Vol. 5628. Lecture Notes in Computer Science. Springer, 2009, pp. 90–107. DOI: 10.1007/978-3-642-03549-4_6. URL: https://doi.org/10.1007/978-3-642-03549-4%5C_6 (cit. on pp. 11, 13).

[28] T. Toft. “Sub-linear, Secure Comparison with Two Non-colluding Parties”. In: *Public Key Cryptography - PKC 2011 - 14th International Conference on Practice and Theory in Public Key Cryptography, Taormina, Italy, March 6-9, 2011. Proceedings*. Ed. by D. Catalano, N. Fazio, R. Gennaro, A. Nicolosi. Vol. 6571. Lecture Notes in Computer Science. Springer, 2011, pp. 174–191. DOI: 10.1007/978-3-642-19379-8_11. URL: https://doi.org/10.1007/978-3-642-19379-8%5C_11 (cit. on p. 13).

[29] T. Veugen, F. Blom, S. J. A. de Hoogh, Z. Erkin. “Secure Comparison Protocols in the Semi-Honest Model”. In: *IEEE J. Sel. Top. Signal Process.* 9.7 (2015), pp. 1217–1228. DOI: 10.1109/JSTSP.2015.2429117. URL: https://doi.org/10.1109/JSTSP.2015.2429117 (cit. on p. 13).

[30] S. Wagh, D. Gupta, N. Chandran. “SecureNN: 3-Party Secure Computation for Neural Network Training”. In: *Proc. Priv. Enhancing Technol.* 2019.3 (2019), pp. 26–49. DOI: 10.2478/popets-2019-0035. URL: https://doi.org/10.2478/popets-2019-0035 (cit. on p. 13).

[31] A. C. Yao. “Protocols for Secure Computations (Extended Abstract)”. In: *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*. IEEE Computer Society, 1982, pp. 160–164. DOI: 10.1109/SFCS.1982.38. URL: https://doi.org/10.1109/SFCS.1982.38 (cit. on pp. 11, 13).

A Summary in German

Multiparty Computation (MPC) bietet die Möglichkeit, verschiedene Funktionen auf privaten Eingaben von mehreren Individuen zu berechnen, ohne die privaten Eingaben oder Ausgaben zu veröffentlichen. In den letzten Jahren haben sich verschiedene Forschungsarbeiten mit diesem Thema beschäftigt und dabei vor allem arithmetische Operationen, wie Addition und Multiplikation, untersucht. Allerdings spielen in der Praxis häufig auch andere Funktionen eine wichtige Rolle. Insbesondere die Vergleichsoperation ist von großer Bedeutung für bestimmte Anwendungen, wie zum Beispiel Geheimhaltung bei Data-Mining, geheimes Bieten bei Auktionen, Privatsphäre bewahrendes maschinelles Lernen oder Geheimhaltung bei linearer Programmierung.

In dieser Arbeit werden sieben verschiedene MPC Protokolle für Vergleichsoperationen von verwandten Arbeiten vorgestellt, analysiert und teilweise implementiert und getestet. MPC Protokolle werden meist in unterschiedliche Kategorien unterteilt, welche die Anwendungsumgebung vorgeben, in denen die Protokolle genutzt werden können. Die Anwendungsumgebung stellt verschiedene Sicherheits- und Angreifermodelle dar. Die Analyse bezieht sich auf die in der Theorie anfallenden Kosten, welche meist in anderen MPC Primitiven (z. B. Additionen/Multiplikationen) angegeben werden. In den Tests wird die Anzahl der Vergleichsoperationen pro Sekunde, sowie die Kommunikation zwischen Individuen pro Vergleichsoperation pro Individuum, berechnet. Dabei werden bis zu 500.000 Vergleichsoperationen pro Protokoll pro Anwendungsumgebung ausgeführt. Fünf der sieben MPC Protokolle für Vergleichsoperationen werden in dem Framework MP-SPDZ [11] implementiert und getestet. Das Framework bietet die Möglichkeit, die Performance von MPC Protokollen in verschiedenen Anwendungsumgebungen zu messen.

Während drei der sieben Protokolle bei den Tests besonders gut abschneiden, zeigen die anderen Protokolle besondere Eigenschaften in ihrem Aufbau oder bei den in der Theorie anfallen Kosten. Durch die besonderen Eigenschaften sind die jeweiligen Protokolle ebenfalls interessant für die Praxis. Die Tests werden sowohl für die vollständigen Protokolle für Vergleichsoperationen, wie auch für Protokollversionen, in denen die Vorberechnungen extrahiert wurden, durchgeführt. Es zeigt sich deutlich, dass eine striktere Anwendungsumgebung, mit einem stärkeren Sicherheits- und Angreifermodell, die Vorberechnungen der Protokolle sehr stark verlangsamt, während die Protokollversionen ohne Vorberechnungen nur leicht negativ beeinflusst werden. Eine striktere Anwendungsumgebung sollte also nur in Erwägung gezogen werden, wenn es möglich ist, die Vorberechnungen separat auszuführen.

Die Ergebnisse dieser Arbeit ermöglichen es, schnelle und einfache Entscheidungen bei der Auswahl des zu implementierenden MPC Protokolls für Vergleichsoperationen für bestimmte Programme, unter Berücksichtigung der Anwendungsumgebung, zu treffen.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature