Privacy-Preserving Collaboration in Network Security

Dilip Many

Master Thesis, 02.03.2009 - 01.09.2009

Advisors: Martin Burkhart, Mario Strasser
Supervisor: Prof. Bernhard Plattner

Computer Engineering and Networks Laboratory (TIK)
Information Technology and Electrical Engineering Department
Swiss Federal Institute of Technology Zurich

Abstract

It is well known that to defend against network security threats having acccess to multi-domain network traffic data is a big help. Nevertheless companies are reluctant to share their network data. SEPIA offers various MPC protocols to solve specific problems in network security.

The first part of the master thesis was to do a literature study to get familiar with the field of network monitoring and anomaly detection and identify potential SEPIA extensions. It was concluded that a weighted set intersection MPC protocol[1] was the most useful SEPIA extension to do. In the second part of the master thesis the protocol was developed and in the third part the protocol was actually implemented. While doing that a package of very helpful classes emerged (the protocol primitives[2]) which allows to access and use basic operations in a simple and intuitive way. In the fourth part the implemented protocol was evaluated for performance. The protocol performed very well, e.g: for 25 peers, 3 privacy peers and 30 (24-bit) features per peer the protocol took less than 3 minutes.

[1]see chapter 3.1 for a description of the protocol
[2]see chapter 4.1

Zusammenfassung

Es ist allgemein bekannt, dass bei der Verteidigung gegen Netzwerkangriffe es von grossem Nutzen ist, wenn man nicht nur Netzwerkdaten aus dem eigenen Netzwerk hat, sondern auch von anderen Netzwerken. Trotzdem teilen Firmen nur sehr selten ihre Netzwerkdaten mit anderen Firmen. SEPIA bietet verschiedene MPC Protokolle um spezifische Probleme im Bereich der Netzwerk Sicherheit zu lösen.

Der erste Teil der Masterarbeit war es, eine Literaturstudie durchzuführen, um mit dem Gebiet des Network Monitoring und der Erkennung von Anomalien vertraut zu werden und mögliche SEPIA-Eweiterungen zu auszumachen. Daraus wurde geschlossen, dass ein Protokoll zu Berechnung der gewichteten Schnittmenge[3] die sinvollste SEPIA Erweiterung ist. In einem zweiten Teil wurde das Protokoll entwickelt und in einem dritten implementiert. Bei der Implementierung wurden zudem einige sehr nützliche Klassen (die protocol primitives[4]) zur einfachen und intuitiven Nutzung von verschieden Operationen erstellt. Im letzten Teil wurde der Ressourcenverbrauch des Protokolls getestet. Dabei hat sich dieses sehr gut bewährt: z.B. bei 25 peers, 3 privacy peers und 30 (24-Bit) features pro peer brauchte es weniger als 3 Minuten.

[3]siehe Kapitel 3.1 für eine Beschreibung des Protokolls
[4]siehe Kapitel 4.1

Contents

1. Introduction

1.1. Motivation

It is well known that to defend against network security threats having acccess to multi-domain network traffic data is a big help [22]. Nevertheless companies are reluctant to share their network data due to privacy concerns and legal risks or other reasons. Currently the most widely used means to share data are non-disclosure agreements and anonymization of the data. While the former still requires a lot of trust, the latter reduces the usefulness of the data and still requires some trust in the recipients of the data. A more elegant solution might be secure multi-party computations. MPC allows to compute functions on the data securly with minimal trust requirements. The only downside is that MPC is much slower. So far this among other reasons kept people from using MPC. SEPIA shall not only proove that MPC can be pratical but also deliver a library of ready-to-use efficient implementations of MPC protocols.

1.2. Problem Statement

To improve network security, functions shall be computable on distributed data sets without revealing anything about the data except for what is reavealed by the final result. To archieve this goal SEPIA uses MPC techniques.

The goal of this thesis is to get familiar with the field of network monitoring and anomaly detection, design and implement a new MPC protocol in SEPIA and to evaluate the protocol.

2. Results of Literature Study

The first part of the master thesis was to do a literature study to get familiar with the field of network monitoring and anomaly detection and identify potential SEPIA extensions.

2.1. Applications

Below is hierarchy trees showing areas of possible applications of network monitoring. The Following sections contain a detailed hierarchy tree of the areas and describe the applications. The descriptions aren't very detailed and concrete procedures how to use the network traffic traces for the applications are often missing. But the focus here is to give a very broad overview of the application possibilities.

```
│
├─Intrusion Detection ........................................ 2.2
├─Performance Monitoring ..................................... 2.3
│ └─network usage analysis
│     └─user behaviour analysis .............................. 2.3.1
└─Network Analysis ........................................... 2.4
      └─user behaviour analysis .............................. 2.3.1
```

The lists of possible applications are partly from own ideas (occured while reading articles) or explicitly from the articles, research papers or application feature lists I read and referenced in the following sections.

2.2. Intrusion Detection

```
│
└─Intrusion Detection ........................................ 2.2
    ├─identify/monitor for specific (byte) pattern
    ├─identify/monitor for protocol message with specific properties
    ├─detect (DoS) attack
    │ └─monitor amount of TCP SYNs
    ├─detect vulnerability scanners
    ├─detect infected hosts
    ├─forensics: analyze attack after it happened
    │ ├─check if information was stolen during (or after) attack
    │ └─identify source of special packet
    │     └─identify origins of emails with attachments
    └─identify/monitor files transfered via smb, ftp, ...
```

By monitoring the packet or flow data of the network intrusion attempts can be detected. For the detection various metrics can be used: volume metrics (amount of traffic in total,

relative amount of some type of traffic, amount of traffic from/to a specific IP address or port, ...), entropy based metrics [13, 9, 12, 25, 34].
If the full packet data is available packets with a specific byte patterns (signature) can be monitored or protocol messages with specific properties can be detected. The former might be a virus or exploit signature (or sensitive business information, phishing mails, or keywords like "bomb", etc.) the later could be a vulnerability based "signature" [16].
Even if just the flow data is available DoS attacks can be detected by monitoring for a sudden increase in packets for a single destination. Monitoring the amount of TCP SYN packets might be of interest too as a sudden increase might indicate an attack.
Detecting vulnerability scanners might be possible too by detecting changes in destination IP address or port entropy or monitoring for special scan specific packets [30].
The detection and identification of infected hosts might be possible as well [14]. This could be done indirectly by monitoring if the host acts like a scanner. Concretly monitoring the number of failed first-contact connection (packets addressed to host with which the sender has not previously communicated) attempts where a high number of failures could indicate infections is a possibility [35]. Another proposition is to detect an infection by a behavioural change. A host might first act as a server and then as a client. (Given a tuple <src IP, src port, dst IP, dst port, time> to describe a packet. If a packet with properties (a,b,c,d,t1) is followed by packet with (c,?,?,d,t2) with t1<t2 c turned from a server into client.) After Host A sent a packet P to B, B might then start to send P (or a packet similar to P) to other hosts indicating that A and B might be infected. The details of these 2 and another behavioural signature can be read in [15].
Monitoring the network data might be helpful for forensic reasons too. After an attack happened the logs can be checked for any anomaly which can be traced back to the attack and be used in the future to identify and prevent similar attacks. The logs might be helpful to assess if any information was stolen as well [6].
Given access to the network packet data of all nodes on the path from the destination to the source the source of a specific packet (e.g. some attack or some email with malicious attachment) could be traced back [8]. But usually traceback is done with other methods [10, 19]:

- probabilistic packet marking: A packet is chosen at random and the router marks it with its address. If sufficiently many packets are marked by the routers in the path of the packet, the path can be reconstructed.
- ICMP traceback: Every 20000th (or so) packets one is chosen and an ICMP traceback message is generated for it and sent to the destination of the selected packet. The traceback message contains information about the next and previous hop a timestamp, a TTL field and possibly some of the payload of the packet.
- overlay network
- hash-based IP traceback
- controlled flooding: In case of DoS attacks the incoming traffic is monitored and the hops among the suspected path of the attack are flooded one by one. If a change in the attack traffic occurs the suspected hop is probably in the path.
- IP traceback with IPSec: During an attack a security association to one of the routers on the supected path is built. If the packets come through that router they will then be authenticated.

- logging: Routers log some properties of the packets passing through to later on determine if an offending packet passed through them. Starting from the closest hop the path can then hopefully be reconstructed hop-by-hop.

Monitoring the files transfered (via smb, ftp, ...) might be important too [29]. It can help identifying data theft (theft of business information).

2.3. Performance Monitoring

```
└─Performance Monitoring ........................................ 2.3
  ├─network performance tuning
  │ └─traffic shaping
  ├─measure service response time
  ├─measuring inter-domain performance
  └─network usage analysis
    ├─usage forecast
    │ ├─plan service maintenance time
    │ └─capacity planing
    └─user behaviour analysis ................................. 2.3.1
      ├─billing
      ├─check for compliance of user behaviour with policy
      ├─measure amount of internet pages using activeX, java, flash,
      │ javascript, ...
      └─application detection/identification
        ├─identify various application versions
        ├─browser identification
        ├─identify virus traffic by signature and measure amount of
        │ traffic
        └─measure traffic amount generated by each application
          └─analyze amount of BitTorrent traffic
            └─analyze files shared by BitTorrent
```

With the network flow data the performance of the network can be measured. Depending on the measurements the performance can then be tuned by e.g. traffic shaping. (reducing the priority of some type of traffic and increasing the priority of some other kind of traffic)
With network flow data we can measure service response time [4]. (e.g. by measuring the time difference between the request and the response) Another application is the measuring of inter-domain performance [33].
Either with the full packet data or the network flow data the usage of the network can be analyzed (e.g: amount of traffic at different times, amount of traffic during special events). With that information an usage forecast can be done to e.g. plan the ideal service maintenance time (e.g. when the service is used the least) [1], or predict the necessity for another server for load balancing or other networking equipment [2].

2.3.1. User Behaviour Analysis

Depending on the kind of network data available many analyses can be done on user behaviour. Depending on how (volume, service, time, ...) a user uses the network he can be

billed accordingly [5]. The user behaviour can be compared to the usage policy too [36]. Given there is a violation consequences might follow.
Given the full packet data is available the amount of internet pages visited using (potentially harmful) technologies like JavaScript, Flash, Java, ActiveX, ... could be measured as well.
The detection and indentification of the applications generating network traffic is possible too [23]. Although with only the flow data available this can be done only approximately by looking at the source and destination ports. However with the full packet data the packets can be scanned for specific signatures. It might be even possible to indentify the specific versions of the applications generating traffic and therefore identify if a user (e.g. with outdated browser) or a webserver (e.g. outdatet webserver software) is at risk [17]. Some special application traffic which could be very interesting to monitor is the traffic of viruses (or spam). After identifying the applications generating network traffic it can be of interest to chart the relative amount of traffic they are generating. E.g. the amount of BitTorrent traffic might be especially of interest to some industries, even more if the files shared could by identified (and mabye their traffic volume quantified too). (The files shared could at least be estimated from the HTTP GET requests for files with the torrent extension.)

2.4. Network Analysis

```
└─Network Analysis ............................................ 2.4
  ├─create network topology map
  │ ├─identify/monitor ip address to mac mapping
  │ ├─identify/monitor domain to ip address mapping
  │ └─identify main network traffic routes
  │   └─identify potential trouble spots/bottle necks
  │     └─congestion monitoring
  ├─relationship discovery
  ├─detect open ports
  ├─identify major data sources
  ├─measure amount of encrypted traffic
  └─Network Diagnosis ......................................... 2.4.1
    ├─help checking network configuration
    ├─troubleshooting of network problems
    └─network health monitoring
      ├─identify failed applications or services
      └─automatic rerouting
```

With access to the network flow data one can do many network analyses like e.g. create a network topology map [23]. The active network addresses can be identified and based on the routing information (e.g. from routing protocol messages, if available) sent through the network the topology can be reconstructed. The IP to MAC address mapping can be monitored too if the ARP packet data is available. The domain to IP address mappping can be monitored by monitoring the DNS replies (or from the HTTP GET requests containing the host field and the destination IP). (A change of the mapping might be e.g. a server change or a phishing attack) Identification of the main network traffic routes should be possible [21]. After identifying the main traffic routes they can be assessed if there might be a potential bottleneck [21]. These can then be monitored for congestions [3].
Depending on the network flow data relationships between hosts can be identified [23]. By

watching the network flow data open ports can be identified [7]. (At least those which received and then sent traffic.)
With the help of the flow data major data sources can be identified. These might be webservers. In case the data is available the most frequently requested pages could then be identified and cached locally.
One might want to measure the amount of encrypted traffic too.

2.4.1. Network Diagnosis

Given the network packet or flow data a couple of things can be done regarding network diagnosis. Network packet data (maybe even flow data) might help checking the network configuration [3] and troubleshoot some other network related problems.
In the area of network health monitoring by monitoring the network flow data one might detect failed applications and services [1]. (E.g. by monitoring if after receiving a request on a known port there is an answer or not.) Based on the network status traffic might be automatically rerouted (e.g. in case of congestion or link failure).

2.5. Applications and SEPIA

This section describes how the different applications described above could benefit from the access to the network data of various companies/institutions instead of just the local one and how SEPIA could be extended for the applications.
For many applications mentioned in the previous sections there is no need and no apparent benefit to collaborate with others. For the following applications the only benefit is to compare to others and get a total sum (statistical purposes):

1. identify/monitor for specific (byte) pattern
2. identify/monitor for protocol message with specific properties
3. monitor amount of TCP SYNs, measure service response time
4. measure amount of internet pages using activeX, java, flash, javascript, ...
5. identify virus traffic by signature and measure amount of traffic
6. measure traffic amount generated by each application
7. analyze amount of BitTorrent traffic
8. measure amount of encrypted traffic

For the following applications collaboration will lead to a more complete picture. But they aren't considered appropriate for a SEPIA extension (as the output itself comprised sensitive data): create network topology map, identify/monitor ip address to mac mapping, identify/-monitor domain to ip address mapping.
To identify the source of some special packet or of an email collaboration with others increases the chance of identifying the effective source. The receiver might ask the others if and when they have seen a specific packet to determine the potential source. In this case SEPIA is probably not of any help.
Measuring inter-domain performance [33] can't be done without collaboration. SEPIA might be of help but it wasn't further investigated.

For analyzing the files shared using BitTorrent collaborating with others we can compute the total count a file was downloaded, the distribution of files downloaded (especially the top 10), the total amount of traffic per file, the relative amount of different file types (e.g: music, videos, games, text, other). These computations could be done with the additive protocol in SEPIA at least for a reasonable amount of files (not for all that exists as that obviously would be too many). The top 10 computation would call for a "heavy-hitter" (top-k) protocol implementation (if the other files shall remain secret).
Collaborating with others we don't just know the major data source of the local network but can compute the overall major data sources. In case the suspects shall remain secret unless they are among the e.g: top 10 SEPIA could be of help with an extension to compute the heavy-hitters. Computing the common major data sources could be done with an extension for set intersection.
Detecting vulnerability scanners is easier when collaborating with others. If the attack isn't restricted to the local network the additional data from others might help to detect the scans earlier. Computing destination port or IP address entropies for this matter is already possible in SEPIA. With a set intersection protocol the common IP source addresses of the offending packets could be identified.
Detection of infected hosts can be easier together when it's done by searching for scanning behaviour (see above). The two changes in behaviour that might help with detection of an infection described in section 2.2 can be done locally.

2.5.1. Correlation of Security Alerts

There is an interest to correlate alerts from different sites for better detection of attacks (among other reasons). However this undertaking might face several problems: disclosure of sensitive information (could be solved by anonymizing data at the cost of data usability), corrupt alert repository, alert flooding, misuse of data from repository by malicious users, vulnerability of repositories against attacks (e.g: probe-request attack) [27] Using Secure Multi-Party Computation would help with keeping sensitive data private, decreases the risk from corrupt repositories and limit the misuse possibilites of malicious participants.
Katti et al. state in their paper [22] that Correlated attacks are prevalent in the Internet: 20% of the offending IP sources attack multiple networks (within a few minutes) and are responsible for 40% of the total alerts (in their dataset). Collaborating with all IDSs isn't necessary. Collaborating with about 4-6 IDSs with which attacks highly correlate is sufficient to detect 95% of the attacks detected by collaborating with them all but only generates about 0.3% of the traffic. Additionally it speeds up blacklisting of about 75% of the common attackers. These collaboration groups are said to not change during 1-3 months. But just randomly collaborating with 4-6 IDSs wouldn't help with attack detection. (Discovering collaboration groups only requires attack time and offending source IPs.)

The authors of [38] roughly divided the correlation approaches into 4 categories:

- similarity based (clustering analysis between alert attributes)
- based on pre-defined attack scenarios (match alerts to scenario templates)
- based on pre- and post-conditions of attacks
- based on multiple nfo sources (correlate alerts from multiple sec. sys.)

In [32] alerts are correlated using approaches that can be put into 3 categories:

- raw packet alert correlation
- frequency-based alert correlation
- n-gram alert correlation

In the category raw payload correlation they mention: String Equality, Longest Common Substring, Longest Common Subsequence, Edit (Levenshtein) Distance. In frequency-based alert correlation byte-frequency distribution and Z-Strings are mentioned. For n-gram alert correlation they used n-gram signatures and n-grams inserted into bloom filters. All techniques worked in their test for correlation of non-polymorphic worms. But for correlating polymorphic worms only longest common subsequence on raw packet data and n-grams seemed promising. (After the true alerts are identified signatures can be generated. The authors state that "Both frequency distributions and Z-Strings can be used as signatures.")
The problem of finding the longest common subsequence for arbitrary many inputs is NP-hard. For 2 (or any other constant number of) inputs it's solvable by dynamic programming. Solving this problem with a secure multi-party protocol would probably be very hard and take way too much time and therefore could hardly be implemented in SEPIA. (This applies for the other mentioned raw packet correlation approaches as well.) To correlate alerts with n-grams the set of suspicious n-grams could be generated by each peer and then the correlation computed together by intersecting the sets of n-grams (if the cardinality is above a certain threshold they are considered similar). Computing this intersection should be possible as a SEPIA extension. (And the computed intersection could then probably be used as signature as well.)
Common threat sources could be identified by an SEPIA extension for set intersection. The top 10 threat source IP addresses could be identified by a heavy-hitter extension.

2.6. Conclusions

The heavy-hitter extension could be used to identify the top files shared in BitTorrent, the major data sources, IP source addresses of the offending packets and threat source IP adresses. The set intersection extension could be used to identify the common major data sources, IP source addresses of the offending packets and threat source IP adresses. Additionally the set intersection extension could help with discovering the collaboration groups mentioned in [22] and correlation of alerts and alert payloads (by computing the intersection of the payload n-grams).
From this we conclude that a set intersection protocol would be more attractive. But if we could instead create a weighted set intersection protocol that would be even better. Then we could treat the set intersection as a special case of the weighted set intersection. But with the weighted set intersection we can additionally constrain the elements which shall be revealed s.t. the weight (number of occurrences or any other score) has to be above a certain threshold. We might for example want to compute the common suspected offensive IP addresses which sent a high amount of TCP SYNs (e.g: above 1200). Instead of revealing those that are suspect to all participants we could just reveal (accuse) those who are suspected by all and have sent a total amount of TCP SYNs above the threshold. In case participant A sees 1000 TCP SNYs from one IP address that might be suspicious from A's perspective. But all others (B-F) see only 1-5 TCP SYNs, therefore the total is <1025 and might not be considered offensive in the end. With the weighing this suspect can then be protected from (false) accusations.
From the weighted set intersection protocol it might be easy to create a heavy hitter protocol too. (By executing the weighted set intersection protocol and cleverly adapting the threshold

till the amount of elements revealed is as desired.) Therefore the most beneficial SEPIA extension is a protocol for weighted set intersection.

The 3 most practical and versatile SEPIA extensions are:

1. weighted set intersection
2. set intersection
3. heavy hitter identification

After carefully going over the results of the literature study we decided that the weighted set intersection protocol shall be implemented.

3. Weighted Set Intersection Protocol Design

This chapter first defines the weighted set intersection protocol (WSIP), describes some preliminaries and the finished protocol design with a (theoretical) analysis of it's security and efficiency.

In order to design the protocol presented later on in this chapter many different ideas were developed. They are described in the appendix A. First, a simple protocol was developed but abandoned at an early stage due to serious security flaws. Then the protocol using Fermat's little theorem was developed (presented in this chapter). Additionally a protocol using polynomials for set encoding and a protocol using GF(2) Shamir shares were developed. Reading the chapter A for details might be well worth the time.

3.1. Weighted Set Intersection Protocol Definition

Let there be a feature set F (e.g. IP addresses, ports, ...) of size $|F|$, a threshold t, $\tilde{t}$ and n participants. With the term weighted set intersection protocol we have a protocol in mind with the following properties:

- Each participant j delivers as input a fixed number s of features $f_i^j \in F$ and a weight/-count w_i^j for each of them.
- For each feature f_i^j let $\omega_i^j = \sum_{k,l:f_k^l=f_i^j} w_k^l$ be the total weight.
- For each feature f_i^j let $\phi_i^j = \sum_{k,l:f_k^l=f_i^j} 1$ be the feature count.
- At the end of the protocol the features f_i^j for which $\omega_i^j \geq t$ and $\phi_i^j \geq \tilde{t}$ shall be revealed together with their ω_i^j, ϕ_i^j and all the peers which had f_i^j in their input sets.
- None of the participants shall learn anything about the features for which $\omega_i^j \geq t$ or $\phi_i^j \geq \tilde{t}$ doesn't hold.

3.2. Related Work

In most of the related work [26, 40, 39] the sets of the peers were encoded as polynomials. A bit different is the work by Kissner and Song [24] which encodes multisets in polynomials. But none of them created a weighted set intersection protocol as described here. They all just did normal set intersection.
In the section A.3 a protocol sketch is described using the idea of encoding the set in polynomials. But it is fairly inefficient and leaks information.

symbol	meaning
n	the number of peers
m	the number of privacy peers
s	the number of features that each peer contributes, i.e. the size of peers' input sets
p	a prime number used as the field size
t	the total weight threshold, e.g. the minimum total weight that a feature needs to have to be revealed
$\tilde{t}$	the intersection threshold, e.g. the minimum number of sets that must contain the feature
f_i^j	the i'th feature contributed by peer j
$[f_i^j]$	the shared i'th feature contributed by peer j
$[f_i^j]_k$	the share k of the shared i'th feature contributed by peer j
w_i^j	the i'th features weight contributed by peer j
max_w	the max. weight value; in case w_i^j has to be upper bounded: $max_w \geq w_i^j$
ω_i^j	the total weight of f_i^j
max_ω	the max. value of ω_i^j
$\hat{\omega}_i^j$	the randomized total weight of f_i^j
ϕ_i^j	the total count of the i'th feature contributed by peer j, e.g the number of sets which contain f_i^j
$\hat{\phi}_i^j$	the randomized total count of feature f_i^j
$r_{i,j}$	a random value for which $r_{i,j} > 0$ holds
max_r	the max. random value; in case $r_{i,j}$ has to be upper bounded: $max_r \geq r_{i,j}$
$\bar{r}_{k,l}$	a random value for which $\bar{r}_{k,l} \neq 0$ holds

Table 3.1.: Notation used to describe protocols.

3.3. Security Model

Peers are participants in the protocol that deliver the input data. They can be actively malicious, i.e. they can deliver any input they want and even deviate from the protocol. Privace peers are defined as participants in the protocol which do all the computations. They are semi-honest, i.e. they can try to gain as much information as possible without deviating from the protocol. To gain information they can also collaborate with other privacy peers or peers. But they must not deviate from the protocol nor deliver wrong computation results.

3.4. Preliminaries

This chapter heavily relies on the notation explained in table 3.1. Most importantly $[x]$ denotes that the value x is a Shamir shared secret. But the reader is advised to go through the entire notation section.
The rest of this section explains some subprotocols that are used in the weighted set intersection protocol.

3.4.1. Shamir Secret Sharing

The protocol relies on Shamir shares [37] to share secrets. Let F be a finite field and $s \in F$ the secret to share. Shamir shares are based on random polynomials $p(x) = s + a_1 * x + a_2 * x^2 + ... + a_k * x^k$ with the property $p(0) = s$. The coefficients a_i are random elements of the finite field F.
Let $\alpha_i \neq 0$ be fixed numbers each associated with exactly one privacy peer i (for all subsequent calculations). Then m shares are generated by giving each privacy peer a share $[s]_i = p(\alpha_i)$. To reconstruct the polynomial and the secret $s = p(0)$ at least k shares are required. The privacy peers simply send their share to every other privacy peer. After a privacy peer has received at least k shares he can interpolate (e.g. using Lagrange interpolation) the secret.
To add a publicly known number b to the shared secret $[a]$ the privacy peers can compute locally $[c]_i = [a]_i + b$. To add two shared secrets $[a]$ and $[b]$ the privacy peers can compute locally $[c]_i = [a]_i + [b]_i$. To multiply a publicly known number b to the shared secret $[a]$ the privacy peers can compute locally $[c]_i = [a]_i * b$.
To multiply two shared secrets $[a]$ and $[b]$ the privacy peers compute locally $d_i = [a]_i * [b]_i$ and share d_i. After receiving at least k shares privacy peer i reconstructs $[c]_i = reconstruct([d_1]_i, [d_2]_i, ...)$ (where $c = a * b$) from the shares $[d_j]_i$. (See [20].)

3.4.2. Equality Check

The protocol uses Fermat's little theorem to construct a function which checks two values for equality. It states that in a field GF(p)[1]:

$$c^{p-1} = \begin{cases} 0 & \text{if } c = 0 \\ 1 & \text{if } c \neq 0 \end{cases} \tag{3.1}$$

The equality test function ist constructed as follows[2]:

$$equal([x], [y]) = 1 - ([x] - [y])^{p-1} \tag{3.2}$$

[1] Galois Field of prime order p

[2] In [31] Nishide et al. present a deterministic equality protocol as well. But that protocol uses much more multiplications. See A.2 for details.

3.4.3. Threshold Check

The protocol compares Shamir shared values against a public constant (e.g: $[x] \geq y$). For this purpose it uses the Less-Than protocol described in [31]. This protocol basically just executes the LSB (least significant bit) protocol up to 3 times (depending on the knowledge about the input values) and does some other multiplications.

A simple example: Assume $p > 2 * max(x, y)$. If $x - y \geq 0$ then $p/2 > x - y (mod\ p) \geq 0$. Otherwise $p > x - y (mod\ p) \geq \lceil p/2 \rceil$. Notice that if $x - y \geq 0$ then $2 * (mod\ p)$ is even, otherwise it's odd. (As $2 * (x - y) < p$ is even and if $2 * (x - y) > p$ then $2 * (x - y)(mod\ p)$ is odd as it's reduced by the odd prime p.)
Therefore $[x] \geq y$ can be computed as

$$LSB\left(2 * ([x] - y)\right), \tag{3.3}$$

where $LSB(\cdot)$ is a protocol computing the least significant bit of its argument.

The LSB Protocol

The $LSB([x])$ protocol from [31] in short:

1. The privacy peers generate the bitwise sharing of a random number $[r \in_R Z_p]_B$ and compute the normal shamir shares of the number $[r]$.

2. The privacy peers compute $[c] = [x] + [r]$ and reconstruct c.

3. The privacy peers compute:

$$(c)_0 \oplus [(r)_0] = \begin{cases} [(r)_0] & \text{if } (c)_0 = 0 \\ 1 - [(r)_0] & \text{if } (c)_0 = 1 \end{cases} \tag{3.4}$$

4. The privacy peers then compute $[(x)_0] = [c <_B r] + ((c)_0 \oplus [(r)_0]) - 2[c <_B r] * ((c)_0 \oplus [(r)_0])$.

Where $(\cdot)_0$ denotes the least significant bit and $\oplus$ denotes the XOR operation.

The Bitwise Less-Than Protocol

The bitwise less-than protocol $[a <_B b]$ from [31] in short:

1. The privacy peers compute in parallel:

$$\forall 0 \leq i \leq l - 1 : [c_i] = [a_i \oplus b_i] = [a_i] + [b_i] - 2 * [a_i] * [b_i] \tag{3.5}$$

2. Then the privacy peers compute the Prefix-Or of c: $\forall 0 \leq i \leq l - 1 : [d_i] = \bigvee_{j=i}^{l-1} [c_j]$

3. The privacy peers compute $[e_{l-1}] = [d_{l-1}]$ and:

$$\forall 0 \leq i \leq l - 2 : [e_i] = [d_i] - [d_{i+1}] \tag{3.6}$$

4. At last the privacy peers compute in parallel:

$$[a <_B b] = \sum_{i=0}^{l-1} ([e_i] * [b_i]) \tag{3.7}$$

The Prefix-Or in step 2 can be computed with the protocol presented in [31], the protocol for PrefixSum in [18], page 62 adapted for Prefix-Or or by simply computing the result bit-by-bit (from bit $l-1$ to 0):

$$[p_{l-1}] = [c_{l-1}] \tag{3.8}$$

$$\bigvee_{j=i}^{l-1} [c_j] = [p_i] = [c_i] \oplus [p_{i+1}] \tag{3.9}$$

The Small Intervall Test Protocol

To check if $\tilde{t} \leq [\phi_k^l]$ a polynomial based threshold check is used as it is more efficient (assuming that $\tilde{t}$ is "small"):

$$[c_{\phi_k^l}] = \tilde{t} \leq [\phi_k^l] = ([\phi_k^l] - \tilde{t}) * ([\phi_k^l] - \tilde{t} - 1) * ([\phi_k^l] - \tilde{t} - 2) * ... * ([\phi_k^l] - n + 1) * ([\phi_k^l] - n) \tag{3.10}$$

See subsection A.2.2 for details.

3.5. The Weighted Set Intersection Protocol

Below is the complete weighted set intersection protocol as it is implemented in SEPIA. The Shamir shares are all in GF(p).

1. All peers and privacy peers have to agree on F, max_w [3], s, a prime $p > max(|F|, 2*m, 2*n*max_w)$ and on the thresholds t, $\tilde{t}$. Additionally they have to agree if the optional steps (3, 4) are done or not.

2. Each peer j selects s features ($f_i^j, 1 \leq i \leq s$) and the corresponding weights that he wants to use in the protocol. He then creates for each f_i^j the shamir shares $s[f_i^j]_1, s[f_i^j]_2, ..., s[f_i^j]_m$ and $s[w_i^j]_1, s[w_i^j]_2, ..., s[w_i^j]_m$ and sends them to the privacy peers.

3. Optional: The privacy peers check each set of features contributed by the peers for duplicates:
   For each peer j the privacy peers compute in parallel:

   $$\forall i, k = [1, s], i \neq k : equal([f_i^j], [f_k^j]), \tag{3.11}$$

   If any of the $equal([f_i^j], [f_k^j]) = 1$ peer j is disqualified.

4. Optional: The privacy peers check that the weights contributed by the peers don't exceed max_w:
   Using the Less-Than protocol the privacy peers compute for each peer j:

   $$c_{w_i^j} = [w_i^j] < max_w \tag{3.12}$$

   If any of the $c_{w_i^j} = 0$ peer j is disqualified. (The LSB protocol can't be used at this point as $w_i^j < p/2$ might not hold.)

5. The privacy peers then compute

   $$\forall i, k = [1, s] : \forall j = [1, n - \tilde{t} + 1] : \forall l = [1, n], l \neq j : [equal([f_i^j], [f_k^l])] = 1 - \left([f_i^j] - [f_k^l]\right)^{p-1}, \tag{3.13}$$

[3]The max_ω used in the previous chapter is implicitly defined here as $max_\omega = n * max_w$

where obviously $equal([f_i^j],[f_k^l]) = equal([f_k^l],[f_i^j])$ (and only one is computed) and

$$equal([f_i^j],[f_k^j]) \begin{cases} 0 & \text{if } i \neq k \\ 1 & \text{if } i = k \end{cases} \tag{3.14}$$

(and none of these cases have to be computed).

6. For each feature f_k^l with $l \in [1, n - \tilde{t} + 1]$[4] the privacy peers compute:

$$[\omega_k^l] = \sum_{j=1}^{n} \sum_{i=1}^{s} [equal([f_i^j],[f_k^l])] * [w_i^j] \tag{3.15}$$

$$[\phi_k^l] = \sum_{j=1}^{n} \sum_{i=1}^{s} [equal([f_i^j],[f_k^l])] \tag{3.16}$$

7. For each feature f_k^l with $l \in [1, n - \tilde{t} + 1]$ the privacy peers compute:

$$[c_{\omega_k^l}] = t \leq [\omega_k^l] \tag{3.17}$$

$$[c_{\phi_k^l}] = \tilde{t} \leq [\phi_k^l] \tag{3.18}$$

8. For each feature f_k^l with $l \in [1, n - \tilde{t} + 1]$ the privacy peers check if:

$$[c_{\omega_k^l}] \wedge [c_{\phi_k^l}] = 1 \tag{3.19}$$

9. For each feature f_k^l for which the check in step 8 succeeded:
   a) f_k^l, ω_k^l and ϕ_k^l are reconstructed by the privacy peers and sent to all peers
   b) for each peer j the privacy peers compute and reconstruct: $\bigvee_{i=1}^{s}[equal([f_i^j],[f_k^l])]$
   c) If the above term evaluates to 1 then peer j is added to the list of peers which had the feature f_k^l in their input set. After constructing these lists they too are sent to all peers.

   Additionally a list of disqualified peers shall be sent to all peers as well.

3.6. Analysis

3.6.1. Security

The security of the protocol is largely based on the security of the Shamir Secret Sharing Scheme.

[4]see subsection A.2.4 for an explanation

Basic assumptions about Shamir shares:

- Less than k shares of a value shared using (k,m)-Shamir shares reveal nothing about the value.
- Any two Shamir shares of two independent values are completely independent. A shamir share of a value a doesn't reveal anything about a value b. Furthermore if one has k-1 shares of a value b the additional knowledge of k-1 shares of the value a reveals nothing about b given that a and b are independent.

Both assumptions are valid as can be seen from Shamirs article [37]. (The second assumption follows from the randomness of the polynomials.)

Security of the Equality Comparison Subprotocol:

- The equality comparison protocol itself does not reconstruct any intermediary results. It uses the power subprotocol.
- The power protocol itself does not reconstruct any intermediary results. The protocol doesn't use any subprotocols either (except for multiplications).

Security of the Less-Than Subprotocol:

- The less-than protocol itself does not reconstruct any intermediary results. It uses the LSB subprotocol.
- The LSB protocol only reconstructs a random value c. This doesn't give any information about the value to compute the least significant bit of. The LSB protocol uses the bitwise less-than subprotocol.
- The bitwise less-than protocol itself does not reconstruct any intermediary results. It uses a subprotocol to compute the prefix-Or.
- The prefix-Or protocol as it was implemented does not reconstruct any intermediary results. The protocol doesn't use any subprotocols either (except for multiplications).

Security of the Small Intervall Test Subprotocol:

- The small interval test protocol itself does not reconstruct any intermediary results. It uses the equality comparison subprotocol.
- The security of the equality comparison protocol was proved above.

Security of the Weighted Set Intersection Protocol: The security of the weighted set intersection protocol itself is proven by demonstrating how the reconstructed intermediary results (concerning the data of the honest peers) can be derived from the final result which is sent to all honest peers. (Obviously if all reconstructed intermediary results can be derived from the final result the intermediary results can't reveal anything that isn't revealed by the final result anyway. The intermediary results which aren't reconstructed are secure as they are shared using Shamir shares.)

- The results of the duplicates check of the honest peers are all 0. The peers which aren't on the list of disqualified peers are obviously honest. (The results of the dishonest peers can't be reconstructed from the final result. But as they are dishonest we don't care about their data anyway.)
- The results of the weight threshold check of the honest peers are all 1. This can be reconstructed from the final result which contains a list of the disqualified peers.
- (The feature equality comparison results aren't reconstructed.)
- (The total weights and total counts aren't reconstructed.)
- (The results of the total weight and total count threshold checks aren't reconstructed.)
- The results of the combined (total feature count and total weight) threshold checks can be derived from the final result. For each honest peer we can derive how many features of his input set are in the final result and therefore had a combined threshold check result of 1. (As the order of the features can be chosen arbitrarily by the peers we can just randomly select the features for which the result shall be 1.)
- At last the protocol reveals as final result for each feature which is in the weighted intersection the feature (value), total weight, total count and the peers that had the feature in their input set. Additionally a list of the disqualified peers is revealed.

If the duplicates check isn't done, a peer could contribute a feature t-1 times. Then if any other peer contributes the feature too and the total weight is above the threshold the feature will be in the intersection. Therefore the total count threshold check can be manipulated.
If the weight threshold check isn't done, a peer could contribute a feature with a weight $\tilde{t}-1$ (which is propably above max_w). Then if any other peer contributes the feature with a weight of at least 1 and the total count is above the threshold the feature is in the intersection. Therefore the total weight threshold check can be manipulated.
In case that both duplicates check and weight threshold check are ommited, the threshold checks can be nullified by any actively malicious peer.
(If both checks are done, a malicious peer can just change the features/weights of his input and analyze the effect on the result. But this ability is present even if the protocol is executed with an honest third party.)

If the feature duplicates checks (step 3) and the weight threshold checks (step 4) are done and there are at most $q = \lceil m/2 \rceil - 1$ malicious privacy peers the protocol is secure.

3.6.2. Efficiency

All subsequent communication effort is measured relative to the effort required to multiply 2 shared secrets.

1. The communication cost for the configuration agreement step might be around 8. The step can be done in 1 round.
2. The communication cost of sending the shares for the weights and features is $2s$ and 1 round.
3. The communication cost of the duplicates check is $n * s(s-1)/2 * log(p)$ and 1 round.

4. The communication cost of the max_w check is $ns * (2 * LSB + 2)$ and the number of rounds used to compute the LSB+2.

5. The communication cost of computing the equals is $(d(d-1)/2 + d * (n-d)) * s^2 * log(p)$ and $log(p)$ rounds, where $d = n - \tilde{t} + 1$.

6. The communication cost of computing one summand of the weight sum is 1. The communication cost of computing the entire sum for all features is $d * n * s^2$ and 1 round. There are no costs for computing the count sum for each feature.

7. The communication cost of the total weight threshold checks for the selected features is $d * s * LSB$ and the number of rounds used to compute the LSB.
   The communication cost of the total count threshold checks (using the small interval test operation) for the selected features is $d * s * (\tilde{t} + log(p))$ and $log(\tilde{t}) + log(p)$ rounds.

8. The communication cost of computing the AND of the total weight and total count threshold checks for the selected features is $d * s$ and 1 round.

9. a) The communication cost of reconstructing a feature, its total count and weight is about 1 each. Therefore the communication cost of this step is at most $3 * d * s$ and 1 round.
   b) The communication cost to construct the lists of peers which had the features in their input set is at most $d * s * n$ and 1 round.
   c) The communication cost of reporting a feature, its total count, weight and the list of peers is at most $3 + n$. The cost for the list of disqualified peers is at most n. Therefore the communication cost of this step is at most $d * s * (3 + n) + n$ and 1 round.

Total Cost

The total communication costs are about (omitting less important terms): $(d(d-1)/2 + d * (n-d)) * s^2 * log(p)$, $d = (n - \tilde{t} + 1)$
The number of rounds required is: $10 + 2 * LSB + 2 * log(p) + log(\tilde{t})$

Concretely: The way the operations were implemented the communication costs are about $1/2 * (n^2 - \tilde{t}^2) * s^2 * log(p)$ and $26 + 6 * log(p) + log(\tilde{t})$ rounds.

4. Weighted Set Intersection Protocol Implementation

This chapter of the documentation gives an overview over the implementation of the Weighted Set Intersection Protocol, based on the design described in chapter 3 of the previous part.

Instead of writing all the code used for the protocol into the specific weighted set intersection classes, the goal was to put the main building blocks, like the equal and less than, into separate classes which are easily accessible for any protocol. Furthermore the user of these protocol primitives (operations) should not be affected by the internal workings of the operations. These building blocks (operations) were put into the class MpcShamirSharingProtocolPrimitives. When using the first few operations of the class in the weighted set intersection protocol it was noticed that some other functions (to send/receive operations data and to synchronize threads) could be reused independent of the operations which were executed. Therefore these functions were put into the classes MpcShamirSharingProtocolPrimitivesPrivacyPeer and MpcShamirSharingProtocolPrimitivesProtocolPrivacyPeerToPP. By inheriting from these new classes and using the inherited functions the usage of the operations was even further simplified. (See 4.1.2)
As the original MpcShamirSharing class didn't generate shares within the specified Galois field (which is essential to many operations used in the weighted set intersection protocol) this class had to be adapted accordingly. Now all generated shares are elements of the field.

4.1. Protocol Primitives

Usually with local operations one might write the operation call as follows:

```
c = multiply(a,b);
```

Therefore the goal was to have the distributed operations on the Shamir shares as similar as possible:

```
id = multiply(a,b);
while(!isOperationCompleted(id)) {
        send( getDataToSend(id) );
        processReceivedData(id, receiveData());
}
c = getResult(id);
```

(Where id is just a number to reference an unique operation execution.)
The final result (using the helper classes) is very similar:

```
initializeNewOperationSet(operationsCount);
operationIDs[0] = 0;
data[0] = a;
data[1] = b;
primitives.multiply(operationIDs[0], data);

doOperations();

c = primitives.getResult(operationIDs[i]);
```

(primitives is the class instance which offers the operations. Lines 2-5 could be put in a loop to initiate several multiplication operations to be run in parallel. See 4.1.2 for more details.)

4.1.1. Operations Overview

The figure 4.1 gives a nice overview over the currently implemented operations. The table 4.1 lists the operations, their costs in terms of multiplications and their round requirements. Check the JavaDoc for details.

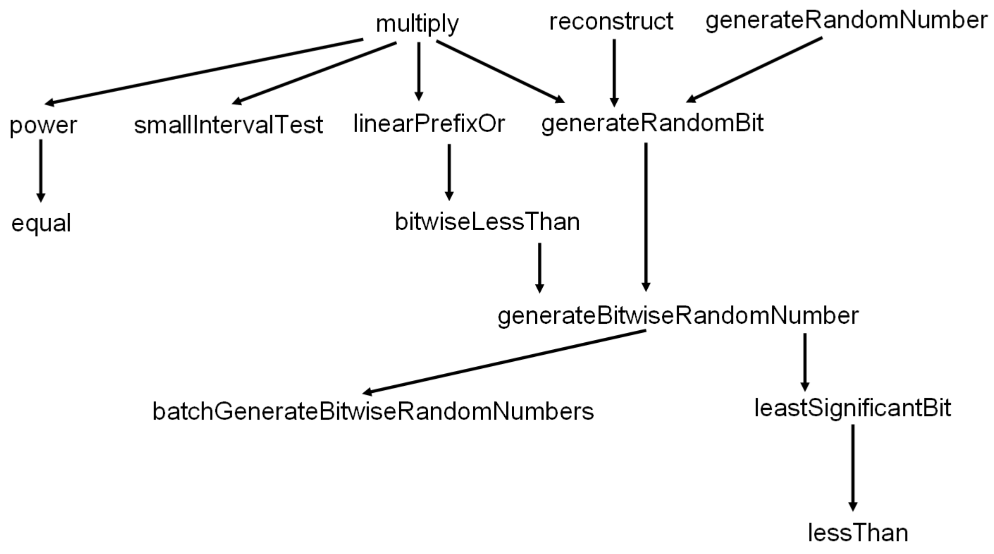

Figure 4.1.: Operations Hierarchy

operation name	multiplications	rounds
reconstruct	1	1
multiply	1	1
power	$\approx 1.5 * log(n)$	$\approx 1.5 * log(n)$
equal	$\approx log(fieldSize) + 2$	$\approx log(fieldSize)$
smallIntervalTest	$\approx u - l + log(fieldSize) + 2$	$\approx log(u - l) + log(fieldSize)$
generateRandomNumber	1	1
generateRandomBit	3	3
linearPrefixOr	$log(fieldSize) - 1$	$log(fieldSize) - 1$
bitwiseLessThan	between $log(fielSize) - 1$ and $3 * log(fielSize) - 1$	$log(fieldSize) + 1$
generateBitwiseRandomNumber	$4 * log(fielSize)$ or $log(fielSize)$	$log(fieldSize) + 5$
batchGenerateBitwiseRandom Numbers	$? * 4 * log(fieldSize)$	$? * (log(fieldSize) + 5)$
leastSignificantBit	$8 * log(fielSize) + 1$	$2 * log(fieldSize) + 8$
lessThan	between $8 * log(fielSize) + 3$ and $24 * log(fielSize) + 5$	$2 * log(fieldSize) + 10$

Table 4.1.: Operations Overview

4.1.2. Usage with Helper Classes

The users of the classes have to make sure that all participants in a computation (privacy peers) do the exact same operations in the same order.
The PrivacyPeer class has to inherit from MpcShamirSharingProtocolPrimitivesPrivacyPeer.
The protocol class for the communication between two privacy peers has to inherit from MpcShamirSharingProtocolPrimitivesProtocolPrivacyPeerToPP.
Before using any operations the numberOfPrivacyPeers variable in the PrivacyPeer class has to be set to the number of privacy peers participating in the computation. In the Protocol class initializeProtocolPrimitives has to be called with the appropriate arguments (see JavaDoc for details).
To start an operation one should create a function in the PrivacyPeer class similar to this one:

```java
public synchronized void startOperations() {
        if(!operationsStarted) {
                operationsStarted = true;
                initializeNewOperationSet(operationsCount);
                operationIDs = new int[operationsCount];
                long[] data = null;
                for(int operationIndex = 0; operationIndex < operationsCount; operationIndex
                    ++) {
                        operationIDs[operationIndex] = operationIndex;
                        data = new long[2];
                        data[0] = initialFeatureShares[featureIndex1];
                        data[1] = initialFeatureShares[featureIndex2];
                        if(!primitives.equal(operationIndex, data)) {
                                logger.log(Level.SEVERE, "equal operation arguments are 
                                    invalid!");
                        }
                }
        }
}
```

Notice:

- The function has to call initializeNewOperationSet with the number of operations to run in parallel.
- initializeNewOperationSet has to be called before allocating the space for the operationIDs.
- The operations must be created exactly once. (Therefore the boolean variable operationsStarted.)
- The operation IDs have to be continous and start with 0. If you have z operations use the IDs from 0 to (z-1).
- The operation IDs have to be stored in the operationIDs variable.
- Create a new data array for each operation otherwise when changing the contents for the next operation you change the data of the previous operation too!

If done so the operations can be executed in the Protocol class (e.g: in the run function) with code similar to the following:

```java
initializeProtocolPrimitives(mpcWeightedSetIntersectionPrivacyPeer, otherPeerInfo);

mpcWeightedSetIntersectionPrivacyPeer.startOperations();
if(!doOperations()) {
        logger.log(Level.INFO, connection.getDescription() + "executing operations failed..."
            );
        sendGoodbye(true);
        return;
}
```

Notice:

- The mpcWeightedSetIntersectionPrivacyPeer variable is a reference to the PrivacyPeer class instance which created this protocol class instance.
- The initializeProtocolPrimitives function has to be called only once before any of the operations are used.

The results of the operations can be retrieved using getResult. But after initializeNewOperationSet was called the results will no longer be available!

Some operations might internally use a specific amount of random variables. If the operations are just called with the standard arguments they will try to generate their random variables themselves. For efficiency reasons and to avoid the potential of the operations failing one might want to generate them in advance. The BatchGenerateBitwiseRandomNumbers operation was created exactly for that purpose. The shares of the bitwise shared random numbers can then be passed to the other operations in the input data. (See the startRandomNumbersPregenerationForWeightsCheck and startWeightsCheck functions in the MpcWeightedSetIntersectionPrivacyPeer class for an example.)

FAQ

- Can operations of different types be processed in the same operation set?
  Yes. But doOperations doesn't return until *all* operations completed.

4.1.3. Usage without Helper Classes

The users of the classes have to make sure that all participants in a computation (privacy peers) do the exact same operations in the same order.
Before creating a new set of operations to run in parallel the initialize function of MpcShamirSharingProtocolPrimitives has to be called. Each operation must be created only once (like when using the helper classes). After creating all the operations processReceivedData has to be called to process the initial data and create the shares to send for the first round (of sending and receiving shares).
All the threads running the protocol between two privacy peers have to run a loop similiar to this one:

```
while(!primitives.areOperationsCompleted(operationIDs)) {
        // send and receive data:
        // to get the data to send use primitives.getDataToSend(ids,
             recipientPrivacyPeerIndex)
        // when data is received use primitives.setReceivedData(operationIDs,
             senderPrivacyPeerIndex, data);

        primitives.processReceivedData(operationIDs);
}
// wait till all threads completed the operations
```

The results of the operations can be retrieved using getResult.

4.1.4. Design

An operation has only three specific functions. One to create the operation and check the intial data (e.g: multiply), doStep to execute a step of the operation and its constructor (e.g: Multiplication).

There are five general public functions: getDataToSend, setReceivedData, processReceived-Data, areOperationsCompleted, getResult.
When one operation uses another internally only the former has an id. The id is only used by the operations which can be accessed externally.
The shares for and from the privacy peers are stored in 2-dimensional arrays, s.t. the opertions could use/compute a variable number of shares per step. The only restriction is that the number of shares for and from each privacy peer in a step is the same. But given that all privacy peers do the exact same thing in the same step this shouldn't be a problem.
As not all privacy peers send their shares before receiving (nor all receive before sending) there are two functions setReceivedData and processReceivedData to set and process the received data (not one combined function as originally intended). With that split the privacy peers can immediately store the data they received for further processing and after receiving and sending they can then call processReceivedData (which only processes the data after all shares for all privacy peers were sent and all shares from all privacy peers received).

Responsibilities of Operation Users

- create operations
- manage operation ids
- manage data sending/receiving as long as operation isn't completed (only if helper classes aren't used)
- get result when operation is completed

Responsibilities of Operations

- manage which data has to be sent
- give data to send (if any)
- store received data
- process received data appropriately (assuming the data is really meant for this operation, which the user of the operation has to check)
- report if operation is completed

4.1.5. Adding New Operations

To add a new operation a new operation class has to be created in the mpc.protocolPrimitives.operations package and a new function to create the operation has to be added to the MpcShamirSharingProtocolPrimitives class.
The function in MpcShamirSharingProtocolPrimitives to create the operation can check the operations input data for validity before calling the constructor of the operation.
The new operation class has to implement the IOperation interface. If the class inherits from GenericOperationState the only thing left to do is implement the constructor for the class and the doStep function.
In general one step consists of whatever has to be done with the intial data or previously received data before sending and receiving the next shares. The final step consists of what has to be done after receiving and sending the last shares to compute the final result of the operation. The number of steps that an operation can have is nearly unlimited (over $2 * 10^9$).

An operation can of course use other operations to do its job. Check out the code of the doStep function of the equal operation for an example.
The operations themselves don't have to care about synchronization. The MpcShamirSharingProtocolPrimitives class takes care of this. Therefore intelligently designed UnitTests should be sufficient to find potential flaws in the newly created operations.

4.2. Performance Optimization

Soon after the weighted set intersection protocol was runnable as unweighted set intersection protocol (on 17.05.2009) the first few performance tests were done. As some of them were quite unsatisfying I improved the code. Some of these improvements were:

- A log message was removed. The creation of the log message encompassed the conversion of integer values to strings and concatenating all these strings. This was done to output all the operation ids of the currently running operations. The message was not very usefull and it took extremly long to construct the messsage. Therefore it was removed.
- In the first version operations which had sub-operations would copy the shares to send from the sub-operations to their array for sending. They would after receiving the shares for the sub-operations copy the shares to the sub-operations. All this copying used lots of time and the same shares existed several times in the memory. Therefore the function (copySharesForPrivacyPeer) was changed s.t. if the shares for sending are to be retrieved the function copies its own shares and then lets the sub-operations directly copy their shares to the destination too. When shares are retrieved the function (copySharesFromPrivacyPeer) retrieves its own shares and then lets the sub-operations get their shares directly too. This way there is no unnecessary copying. Although the functions to set and retrieve the shares are more complicated the code now runs much faster.
- Instead of using a loop with the sleep function call to synchronize the protocol threads a SimpleCountingBarrier class was created and used. This class uses Javas wait and notify functions. The change made the code much faster.
- The memory consumption of the multiplication operation could be reduced by creating a special MultiplicationState for the operation. The overall memory consumption reduction was not as big as expected but still significant. (Maybe the JavaCompiler already optimized the code quite well before the changes...)

The protocol with 3 privacy peers, 10 peers and 50 features per peer took:

time (s)	version information
1043	first version
306	after removing unnecessary log message
195	after reducing memory usage (by removing copying shares from/to sub-operations)
84	final WSIP[1] using SimpleCountingBarrier and many other improvements

[1]this final protocol does many things that the older versions didn't; therefore this comparison is a bit unfair to the final version

4.3. Weighted Set Intersection Protocol Usage

To use WSIP the peers and privacy peers have to be configured consistently and the input data has to be in the appropriate format and place. After this is done WSIP can be run like any other SEPIA protocol.

4.3.1. Configuration

In general WSIP is configured like all other SEPIA protocols. The parameters which directly affect the WSIP behaviour are explained in the table below. Except for the parameter "trafficfilepath" all other parameters have to be the same among all the peers and privacy peers.

Parameter	Description
peers.trafficfilepath	path to directory containg the input data, e.g: ./input/
peers.numberoftimeslots	The (integer) number of timeslots for which the input data is processed per protocol execution.
peers .numberofitemsin-timeslot	This parameter has to be set to 1.
mpc.weightedsetintersection .maxweight	The max. value the weights contributed by the peers can attain.
mpc.weightedsetintersection .maxfeature	The max. value the features contributed by the peers can attain.
mpc.weightedsetintersection .featurecountperpeer	The number of features sent per peer and timeslot.
mpc.weightedsetintersection .mintotalweightthreshold	The min. total weight a feature has to have to be in the intersection.
mpc.weightedsetintersection .minfeaturecountthreshold	The min. total count a feature has to have to be in the intersection.
mpc.weightedsetintersection .featureduplicatescheck	true or false depending on if the feature duplicates check shall be done.
mpc.weightedsetintersection .maxweightcheck	true or false depending on if the max. weight check shall be done or not.
mpc.weightedsetintersection .inputdatatimeout	0 if WSIP shall be terminated after processing all data from all timeslots. Number of milliseconds (> 0) to wait if the peers and privacy peers shall wait for new input data after processing all data from all timeslots to restart WSIP automatically.

Table 4.2.: WSIP parameters

inputdatatimeout

Instead of terminating after processing all the data of all the timeslots, the peers and privacy peers can be instructed to wait a specified amount of time for new input data. After sufficient data for the specified number of timeslots is available WSIP will be restarted.
If the specified amount of time passes without the new input data being ready, WSIP will be terminated. (The privacy peers will actually wait a little longer to give the peers time to send at least a goodbye message in time.)

4.3.2. Input Data Format

The input data files can have any name. They just have to be in the directory specified in the configuration. Each file will be used to generate the input data for one timeslot. (Therefore unless inputdatatimeout=0 there should be at least as many input files as there will be timeslots.) The files will be read and used in lexicographical order.
Each line in the input file can have several columns separated by ";". The first column will be considered to be the feature. The weight is the number of lines which contain the feature.

4.3.3. Output Data Format

The output data is written to the same directory where SEPIA was started from. The files will be named either "sepia.output..." or "Result_...". (The first version in case inputdatatimeout=0 otherwise the second variant applies.)
Each line contains a feature (of the intersection), its total weight, its total count and the list of peer IDs of the peers which had the feature in their input set. These items are separated by a tab. The list of peer IDs is tab-separated too. (The order of the features can be arbitrary.)

4.4. Additional Tools

In the process of implementing the protocol some other neat tools were created.

4.4.1. PrimeNumberGenerator

In order to make the equality comparison operation work as fast as possible, it's essential to have many prime numbers with a as few 1-bits as possible. Ideally, if $x = max(|F|, 2*m, 2*n*max_w)$ we want to have a prime p, s.t. $p > x$, $\lceil log(p) \rceil = \lceil log(x) \rceil$ and p has only two 1-Bits in its bit-representation. That way the exponentiations with p-1 would be as fast as possible.
To that end I wrote the program PrimeNumberGenerator to find as many prime numbers as possible with as few 1-bits as possible. 55 good prime numbers were found (with up to 63 bits). These primes only have between 2 and 4 1-bits!
Some examples of good prime numbers can be seen below:

prime	bits count	bit representation
65536	17	10000000000000001
12884901893	34	1100000000000000000000000000000101
2256266579673089	52	1000000001000001000000000000000000000000000000000001

4.4.2. AlertTestDataGenerator

To generate a larger amount of test data than were possible by writing by hand AlertTestDataGenerator was created. With that program 3 input sets with 20'000 data entries each were created.
The program can generate a set of random features (potentially) common to all data sets. Then it generates for each peer a set of random features. Then for each peer it chooses a given number (e.g. 20'000) of features from either the "common" or "individual" set. The chosen feature is written to the first column, for each other column it chooses just some random value from the specified (for each column) range. (See the JavaDoc of the program for details.)

5. Evaluation on Student-Cluster

5.1. Scenario

The goal of the evaluation was to test how the running time of the weighted set intersection protocol is affected by the number of peers, privacy peers and features per peer. Therefore the protocol was run several times with varaying values for these parameters. By knowing how these parameters affect the performance we can estimate in advance if the protocol is suitable for a given scenario with given parameters and how the parameters can be changed slighlty to make the protocol usable for the scenario.
It was intended to test WSIP with DShield data. Unfortunately the DShield data we received for the evaluation was missing the sender id. Therefore artifical data generated by AlertTest-DataGenerator 4.4.2 was used for the evaluations instead.

5.2. SEPIA Code Modifications and Extensions

As the data used was generated with AlertTestDataGenerator 4.4.2 no changes had to be made to WSIP for the evaluations.

5.3. Setup

The weighted set intersesction protocol was run on the student cluster tardis-c. Due to the fact that all students can access the cluster SEPIA wasn't the only program running on these machines. Most often the machines were busy running matlab jobs (using nearly 100% of the machines ressources). Some machines were actively used by students for browsing or other things (using a varying amounts of the machines ressources). Therefore the accuracy of the results in general is rather low.
The tardis-c computers run GNU/Linux with an Intel Core2 2.40GHz CPU and 2GB RAM. They are connected to a 100MBit/s network.

Each peer had an input file with 20 000 data entries. Each protocol round with a specific parameter set was run 10 times to eliminate noise in the results as much as possible. The optional feature duplicates check and the weights threshold check were executed in all test runs.

The elapsed wall clock time wasn't plotted as it was even more susceptible to the influence to matlab jobs running in parallel on the privacy peers even though it was measured. (The wall clock time decreased even though the user time and the work increased for some configurations...) The user time of the peers was not plotted as it was always around 5s. It increased to 7s if the number of privacy peers was greatly increased. This difference was considered unimportant. The average amount of messages sent by the privacy peers was measured as well but didn't vary (depending on the parameters) as much as the number of bytes sent. The number of bytes sent and received by the privacy peers was always about the same.

The average round times mentioned are in general *without* the time spent generating bitwise shared random numbers. The time spent generating the numbers varied randomly and therefore didn't give any information about the performance of WSIP. The time to generate the numbers was anywhere between 10% and over 50% of the total round time. This generation could probably be improved.

5.4. Dependence on Peers Count and Minimum Feature Count Threshold

5.4.1. Results

The following plots show how the user time of the privacy peers (Fig. 5.1), the average round time (Fig. 5.2), the maximum memory usage of privacy peers (Fig. 5.3) and average data sent per round by privacy peers (Fig. 5.4) depend on the peers count if the min. feature count threshold increases along with the peers count (always half of the peers count).
The different lines represent the results for different privacy peers counts. The features count per peer was set to 50 and 24 bit features were used.
The outlier with 3 privacy peers and 9 peers in the average round time (Fig. 5.2) plot is probably just due to the varaying background load of the machines.
All the plots and lines show a roughly quadratic growth.

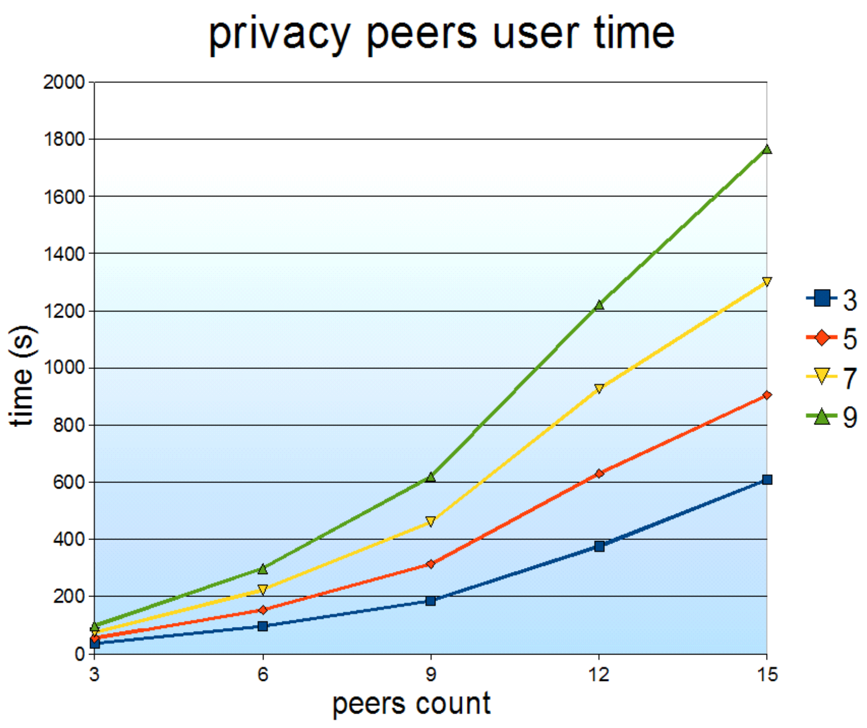

Figure 5.1.: user time of privacy peers depending on the peers count (for different privacy peers count)

The following plots show how the user time of the privacy peers (Fig. 5.5), the average round time (Fig. 5.6), the maximum memory usage of privacy peers (Fig. 5.7) and average

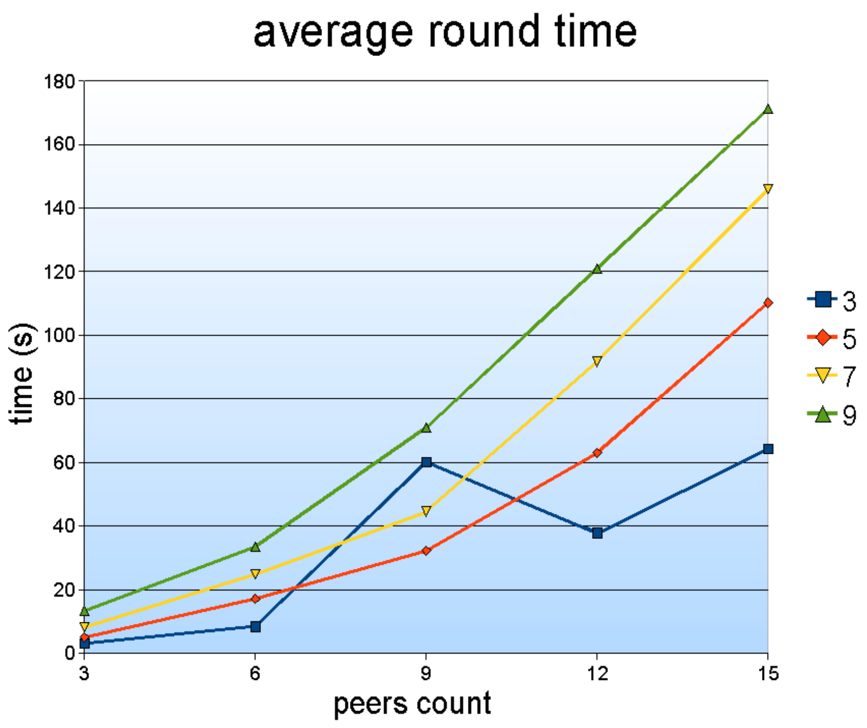

Figure 5.2.: average round time of privacy peers depending on the peers count (for different privacy peers count)

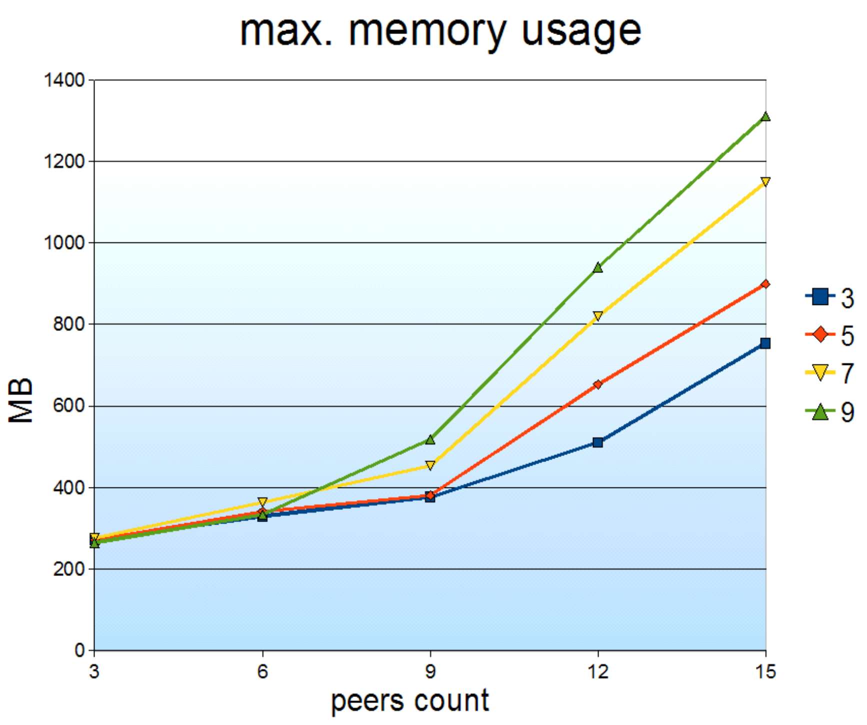

Figure 5.3.: maximum memory usage of privacy peers depending on the peers count (for different privacy peers count)

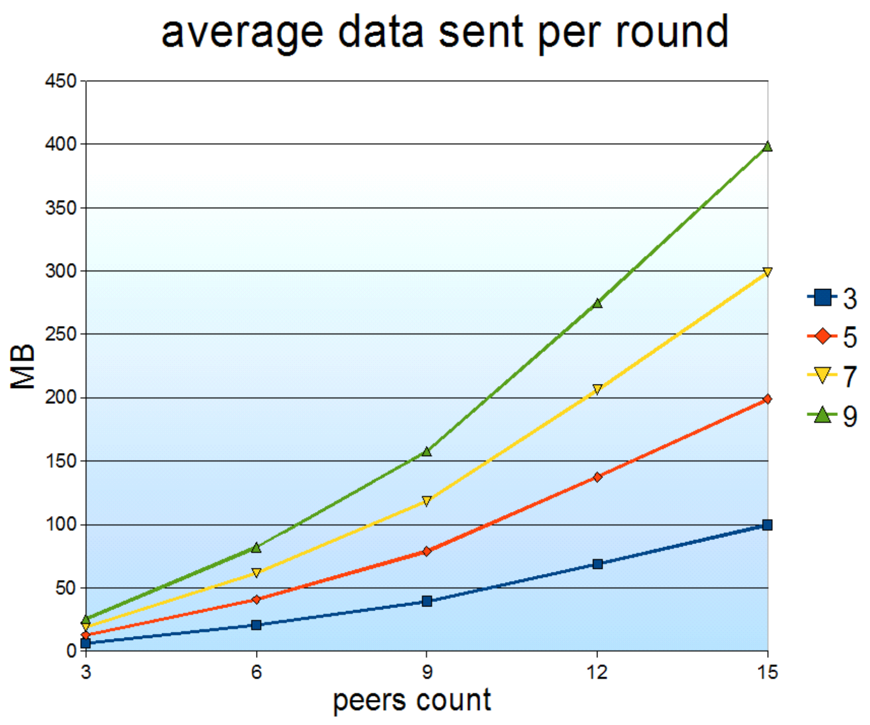

Figure 5.4.: average data sent per round by privacy peers depending on the peers count (for different privacy peers count)

data sent per round by privacy peers (Fig. 5.8) depend on the min. feature count threshold when the peers count is fixed (to 12).
The features count per peer was set to 50, the privacy peer count was 3 and 24 bit features were used.
The user time of the privacy peers (Fig. 5.5) and the average data sent per round by privacy peers (Fig. 5.8) show a roughly quadratic decay. The other two plots roughly show a quadratic decay as well if one ignores the last data point (where the decay decreases).

The following plots show how the user time of the privacy peers (Fig. 5.9), the average round time (Fig. 5.10), the maximum memory usage of privacy peers (Fig. 5.11) and average data sent per round by privacy peers (Fig. 5.12) depend on the peers count when the difference of the min. feature count threshold to the peers count is fixed (to 3).
The features count per peer was set to 50, the privacy peer count was 3 and 24 bit features were used.
The user time of the privacy peers (Fig. 5.9) and the average round time (Fig. 5.10) show a roughly linear growth. The maximum memory usage of privacy peers (Fig. 5.11) seems to grow quadratically. The average data sent per round by privacy peers (Fig. 5.12) shows a perfectly linear growth.
The table 5.1 summarizes the dependency relations of this subsection.

5.4.2. Interpretation

We can conclude that the user time of the privacy peers, the average round time, the maximum memory usage of privacy peers and the average data sent per round by privacy peers depend quadratically on the peers count (if the min. feature count threshold increases along with the

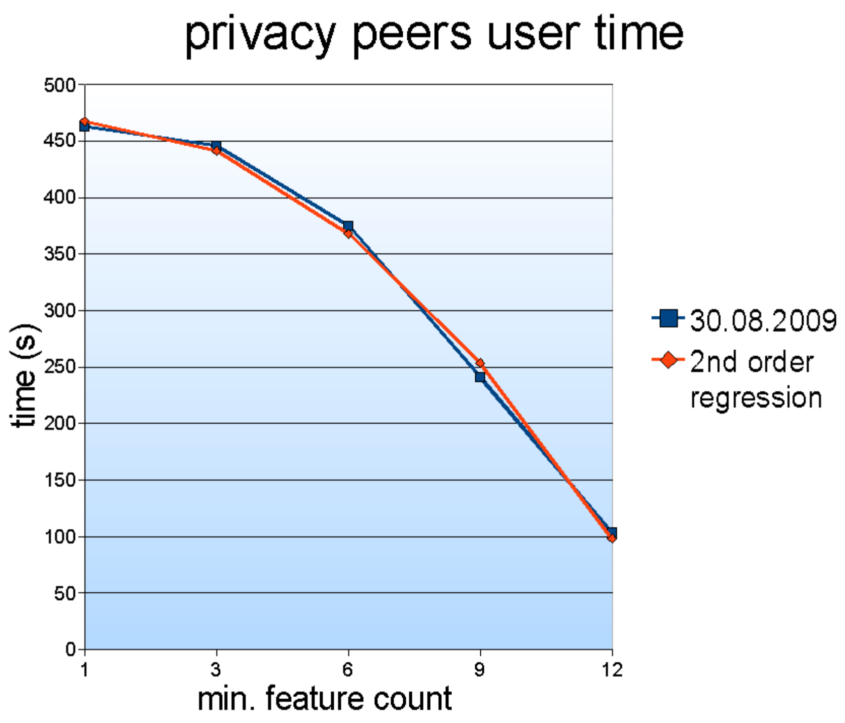

Figure 5.5.: user time of privacy peers depending on the min. feature count threshold (for a fixed peers count)

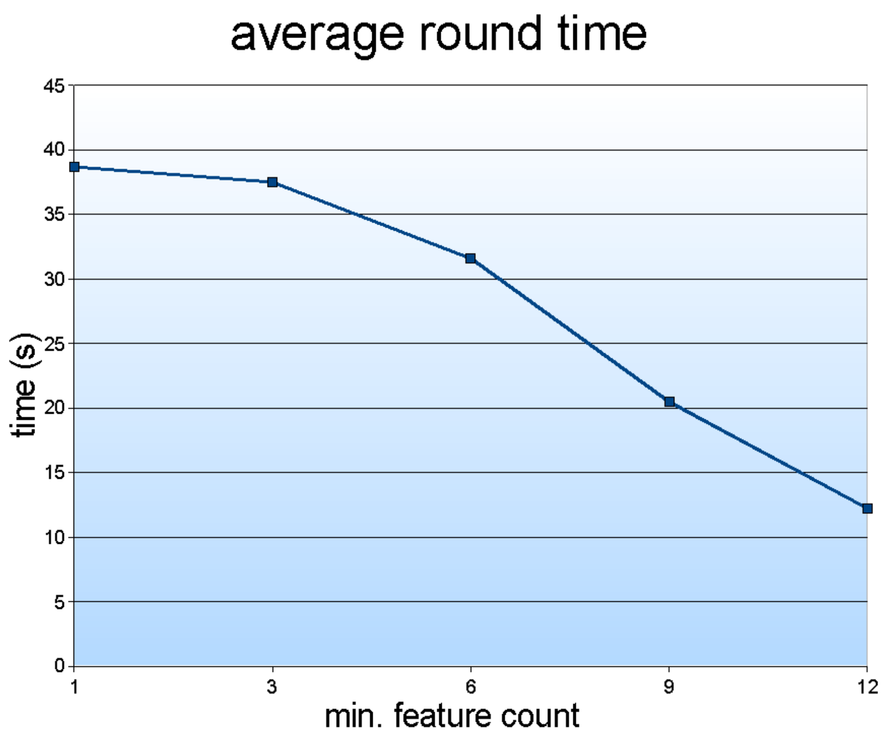

Figure 5.6.: average round time of privacy peers depending on the min. feature count threshold (for a fixed peers count)

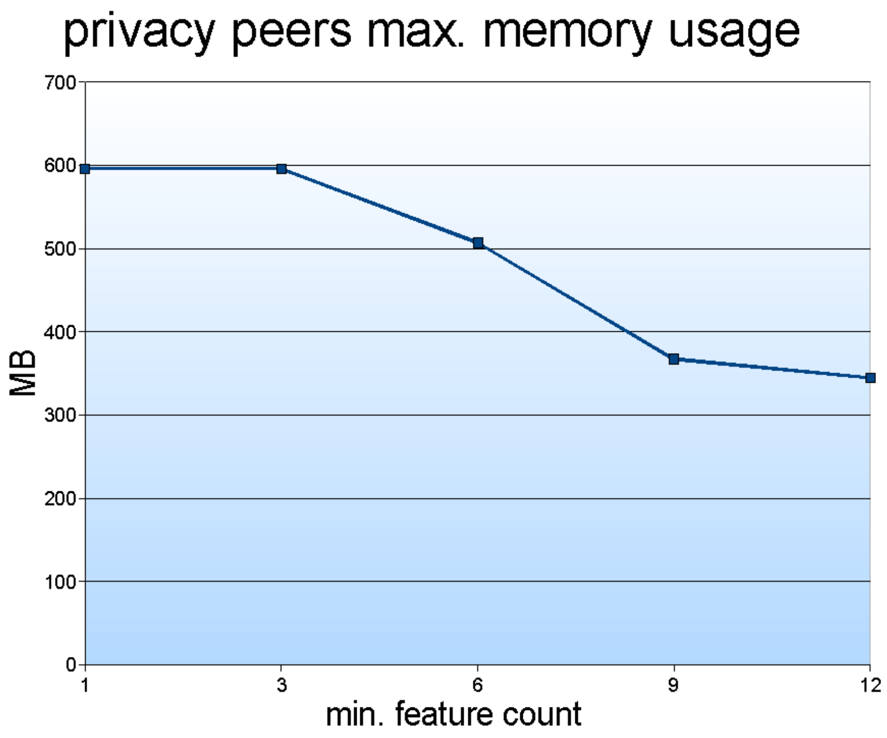

Figure 5.7.: maximum memory usage of privacy peers depending on the min. feature count threshold (for a fixed peers count)

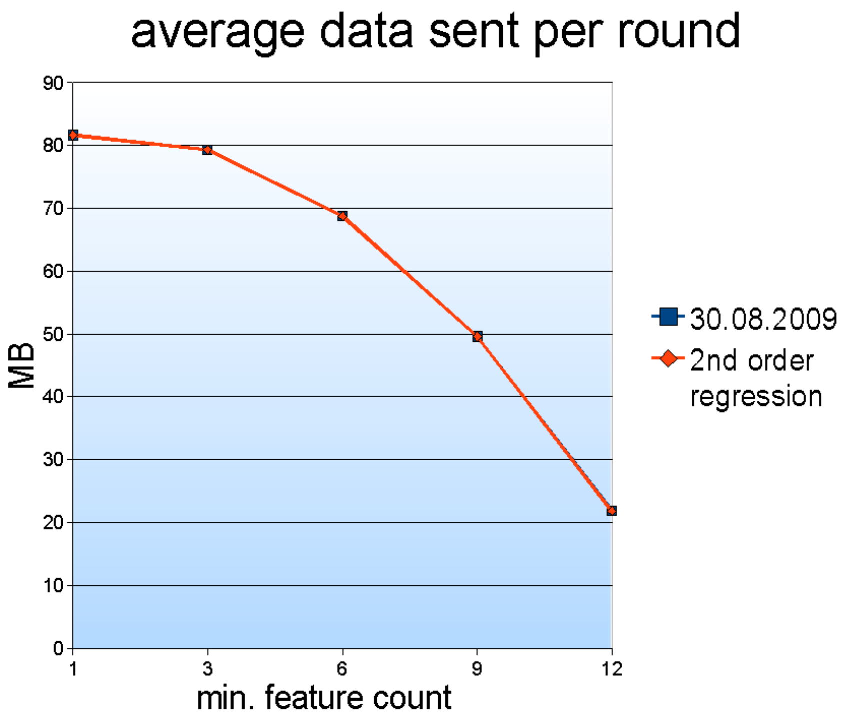

Figure 5.8.: average data sent per round by privacy peers depending on the min. feature count threshold (for a fixed peers count)

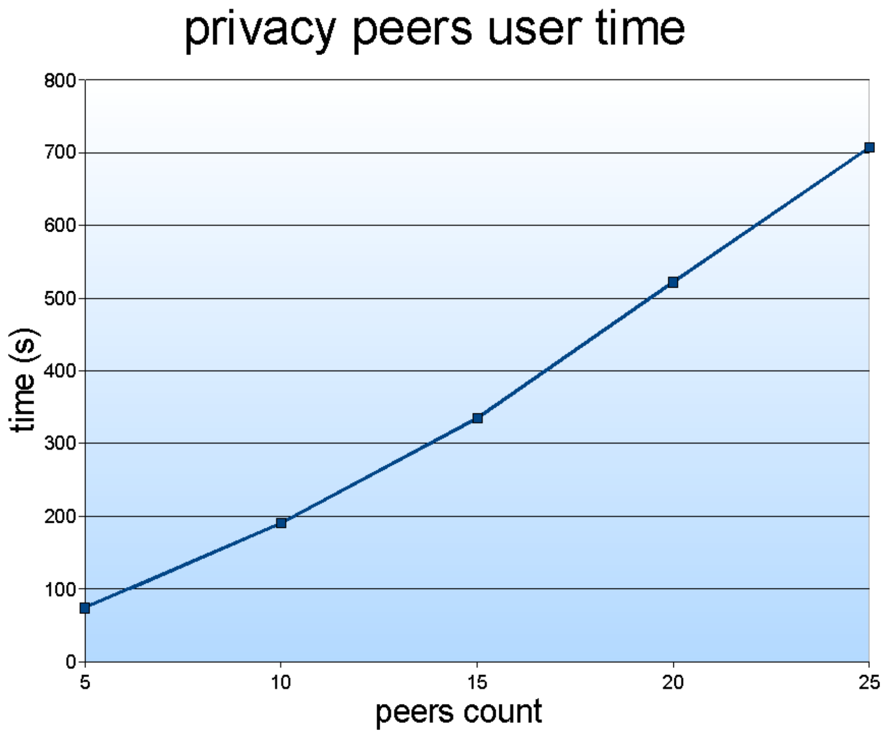

Figure 5.9.: user time of privacy peers depending on the peers count if the difference of the min. feature count threshold to the peers count is fixed

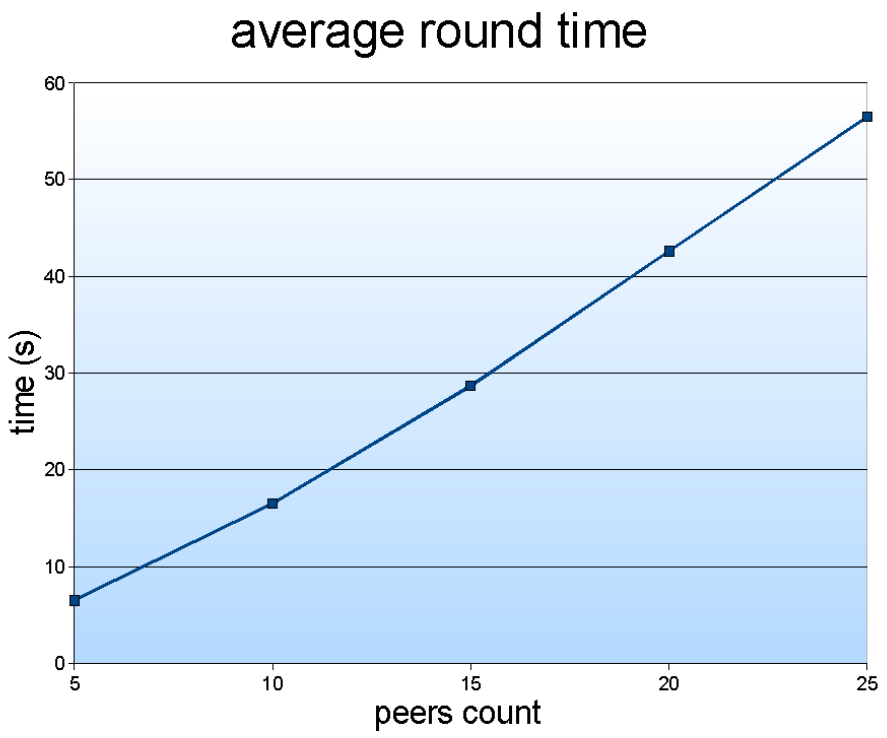

Figure 5.10.: average round time of privacy peers depending on the peers count if the difference of the min. feature count threshold to the peers count is fixed

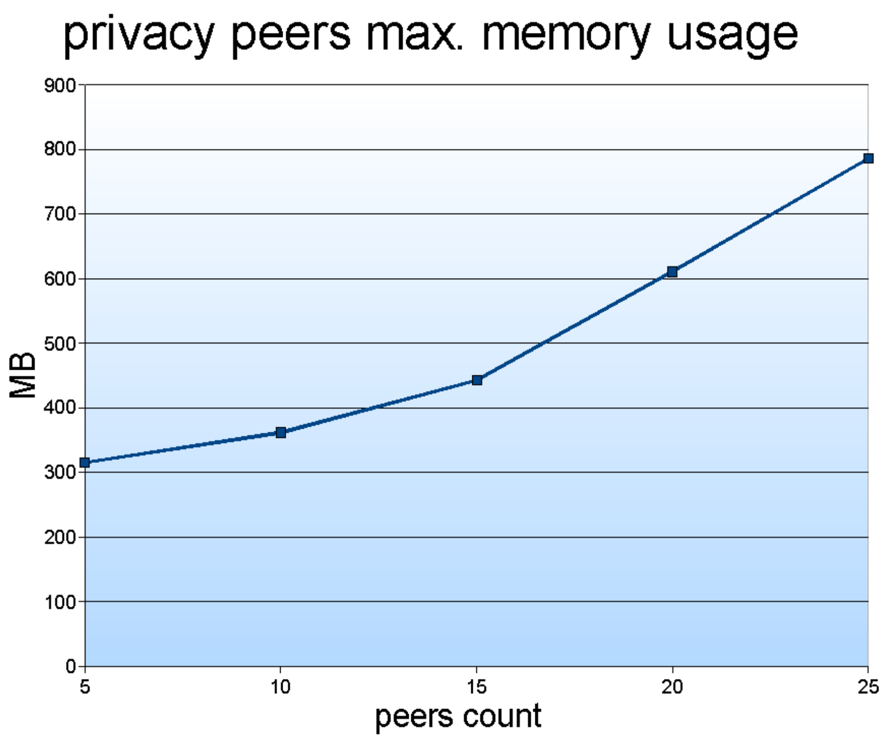

Figure 5.11.: maximum memory usage of privacy peers depending on the peers count if the difference of the min. feature count threshold to the peers count is fixed

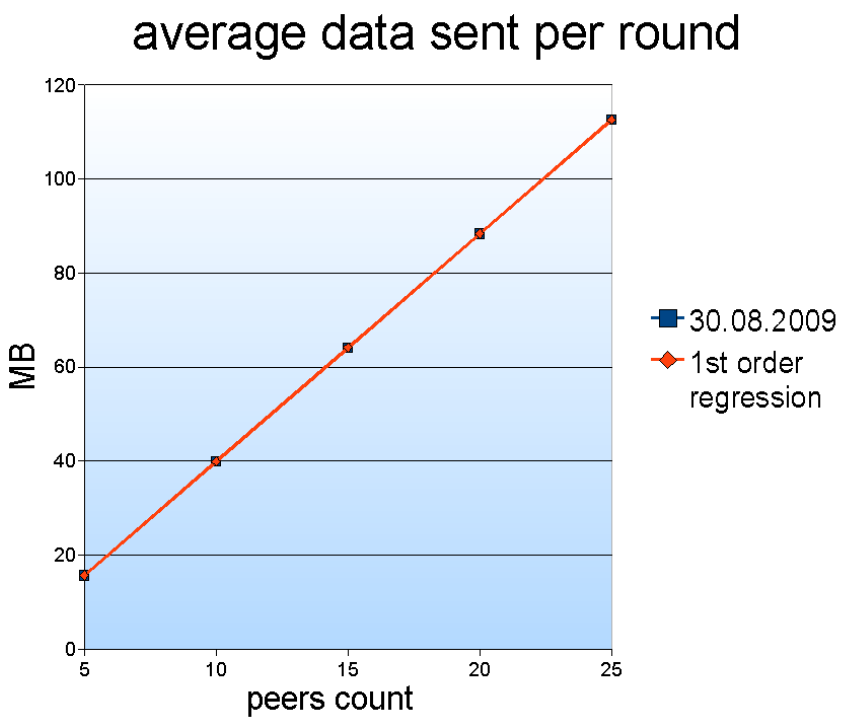

Figure 5.12.: average data sent per round by privacy peers depending on the peers count if the difference of the min. feature count threshold to the peers count is fixed

	peers count and min. feature count increase	min. feature count increase, peers count is fixed	peers count increases, diff. of min. feature count to peers count is fixed
privacy peer user time	quadratic growth	quadratic decay	linear growth
average round time	quadratic growth	quadratic decay	linear growth
privacy peer max. memory usage	quadratic growth	quadratic decay	quadratic growth
privacy peer average data sent per round	quadratic growth	quadratic decay	linear growth

Table 5.1.: Summary of dependency relations for this subsection.

peers count). This was expected from the result of the WSIP efficiency analysis 3.6.2. To verify the quadratic growth, a regression with a 2nd order polynomial was computed for the average round time 5.13 and the average data sent per round 5.14 (for yet another set of measurement data, with 3 privacy peers and 30 features per peer). The regression fits the data very well, which supports the claim of quadratic growth.

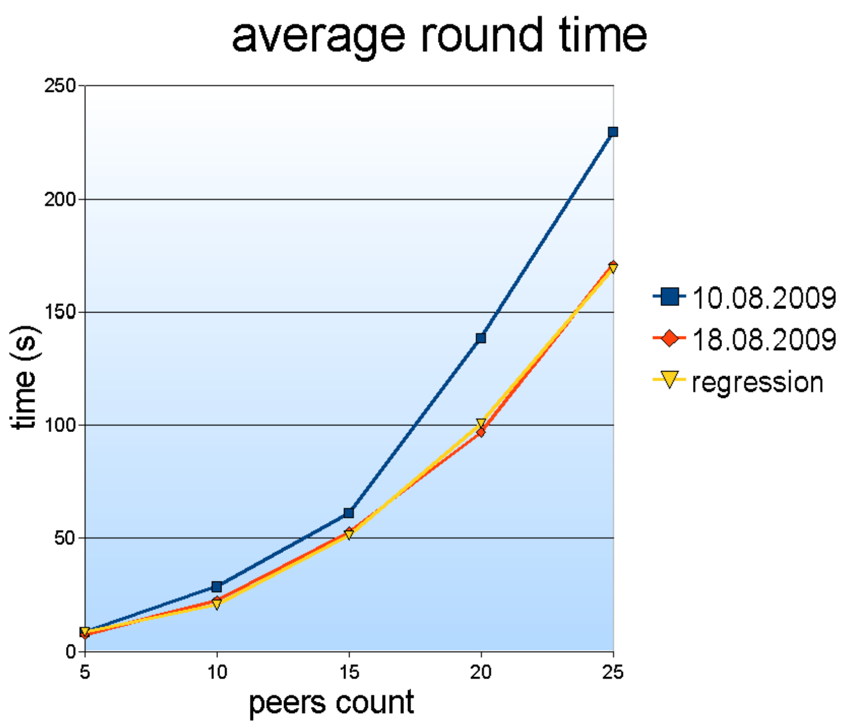

Figure 5.13.: average round time depending on the peers count

If the peers count is fixed while the min. feature count increases, the variables show a quadratic decay. If the difference of the min. feature count to the peers count is fixed and the peers count increases, the user time of the privacy peers (Fig. 5.9), the average round time (Fig. 5.10) and average data sent per round by privacy peers (Fig. 5.12) increase only linearly. (The maximum memory usage of privacy peers (Fig. 5.11) is the only variable which still seems to grow quadratically.)
If a privacy peer has only 1GB of RAM executing WSIP with 15 peers might be hardly possible or even impossible depending on the other parameters.

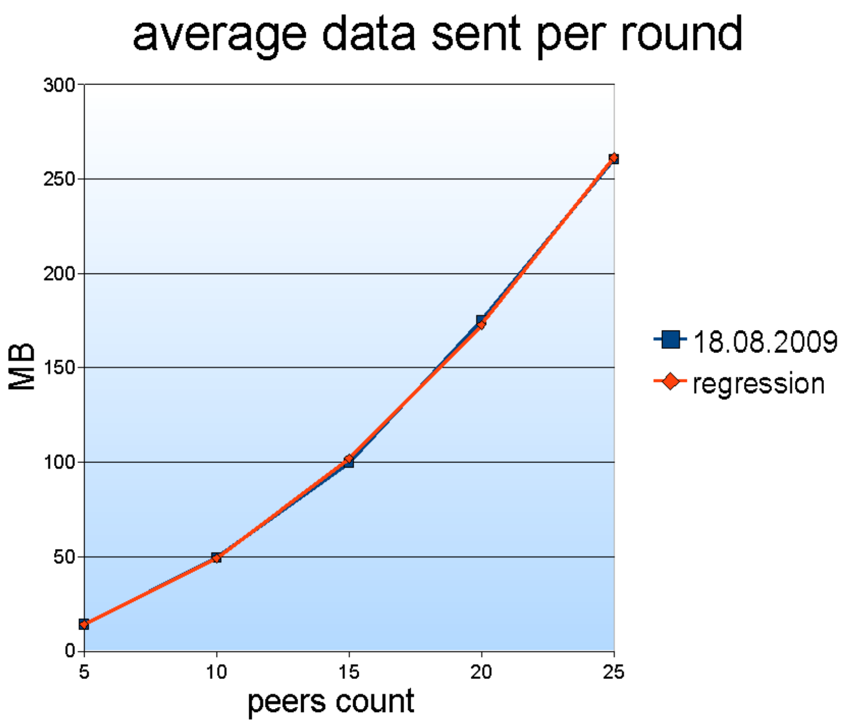

Figure 5.14.: average data sent per round by privacy peers depending on the peers count

5.5. Dependence on Privacy Peers Count

5.5.1. Results

The following plots show how the user time of the privacy peers (Fig. 5.15), the average round time (Fig. 5.16), the maximum memory usage of privacy peers (Fig. 5.17) and average data sent per round by privacy peers (Fig. 5.18) depend on the privacy peers count.
The different lines represent the results for different peers counts. The features count per peer was set to 50 and 24 bit features were used.
The outlier with 3 privacy peers and 9 peers in the average round time (Fig. 5.16) plot is probably just due to the varaying background load of the machines.
All the plots and lines show a roughly linear growth.

5.5.2. Interpretation

While for the first set of test run data we could conclude that the user time of the privacy peers, the average round time, the maximum memory usage of privacy peers and the average data sent per round by privacy peers depend linearly on the privacy peers count. Further test runs with up to 25 privacy peers (and 3 peers and 50 features per peer) showed that the user time of the privacy peers and the average round time 5.19 rather grow quadratically. But the average data sent per round by privacy peers depends linearly on the privacy peers count as shown by 5.20.
If a privacy peer has only 1GB of RAM executing WSIP with 9 privacy peers might be hardly possible or even impossible depending on the other parameters.

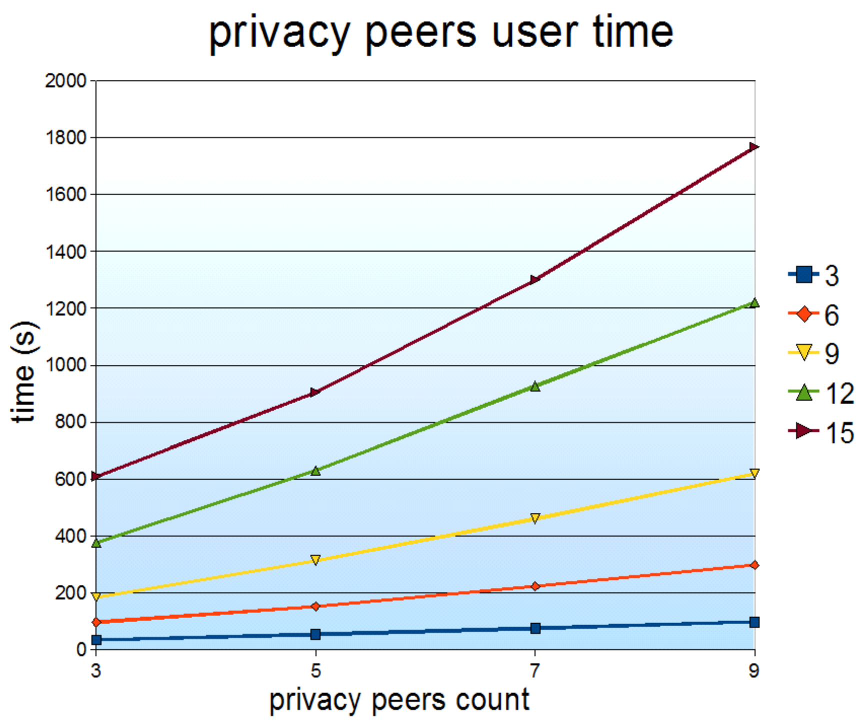

Figure 5.15.: user time of privacy peers depending on the privacy peers count (for different peers count)

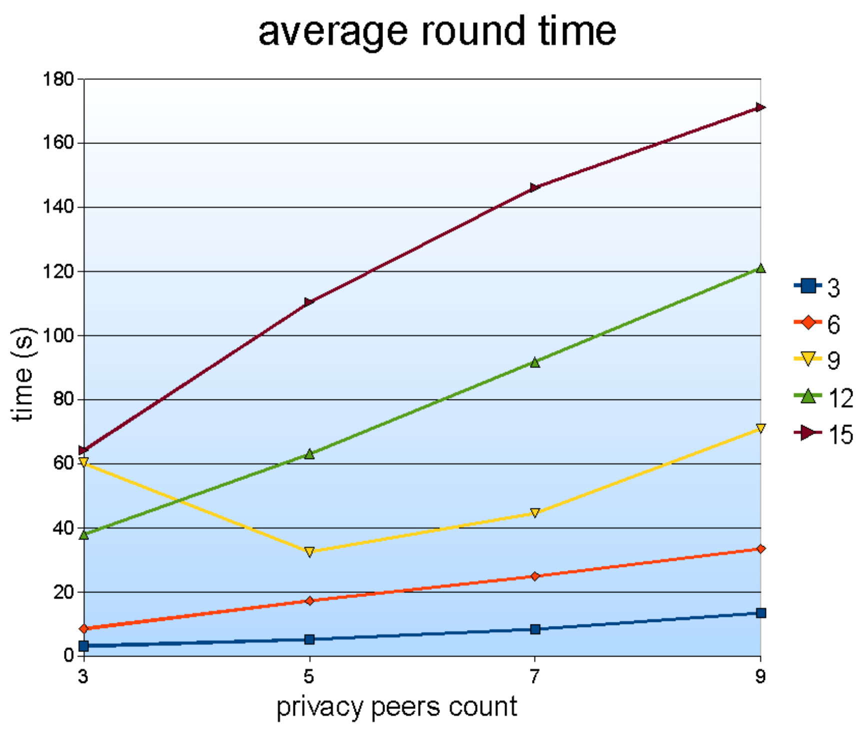

Figure 5.16.: average round time of privacy peers depending on the privacy peers count (for different peers count)

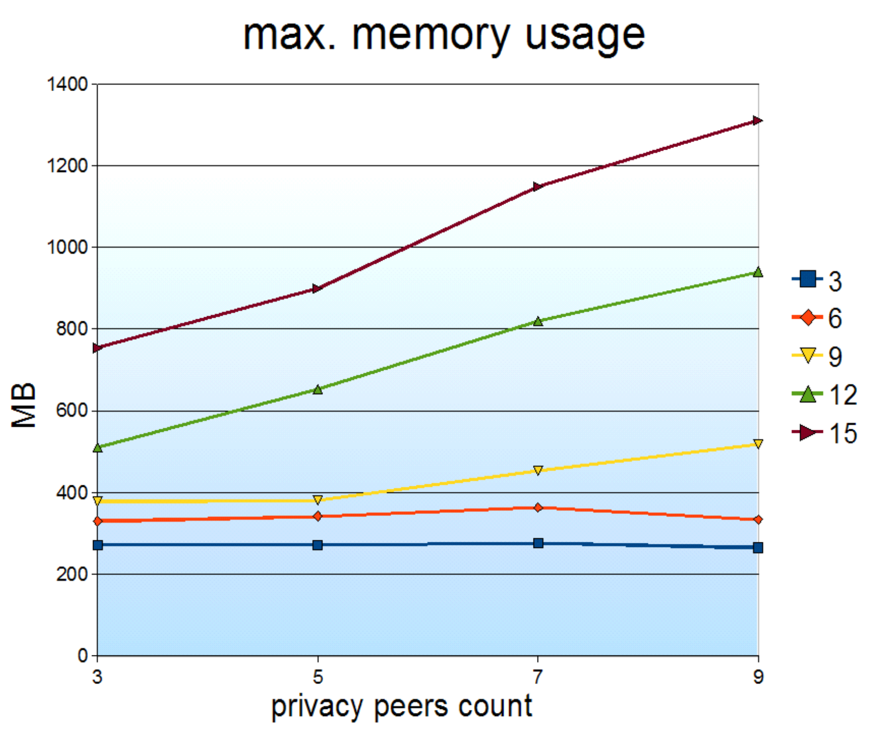

Figure 5.17.: maximum memory usage of privacy peers depending on the privacy peers count (for different peers count)

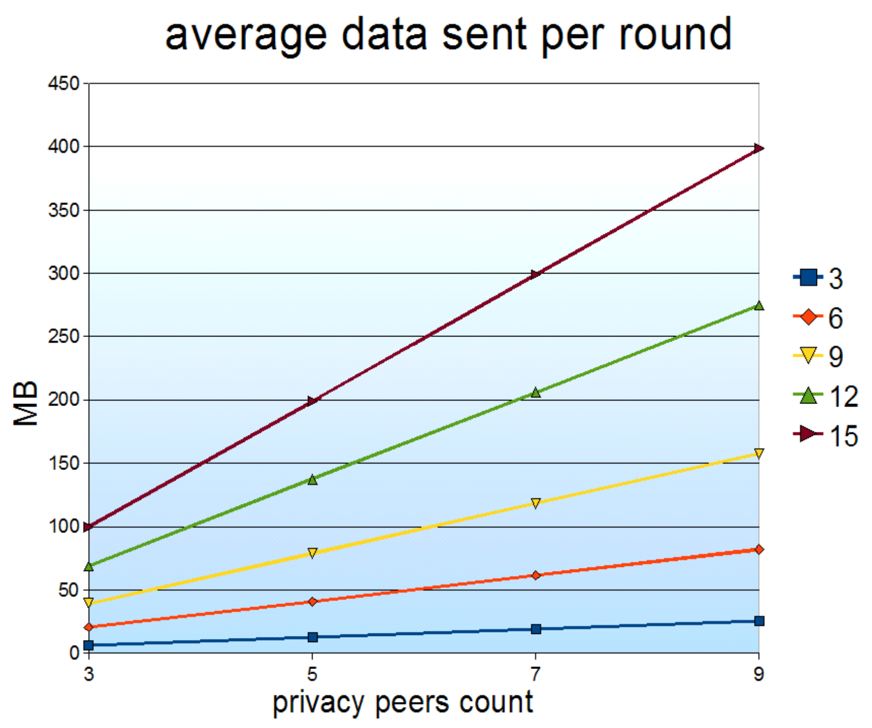

Figure 5.18.: average data sent per round by privacy peers depending on the privacy peers count (for different peers count)

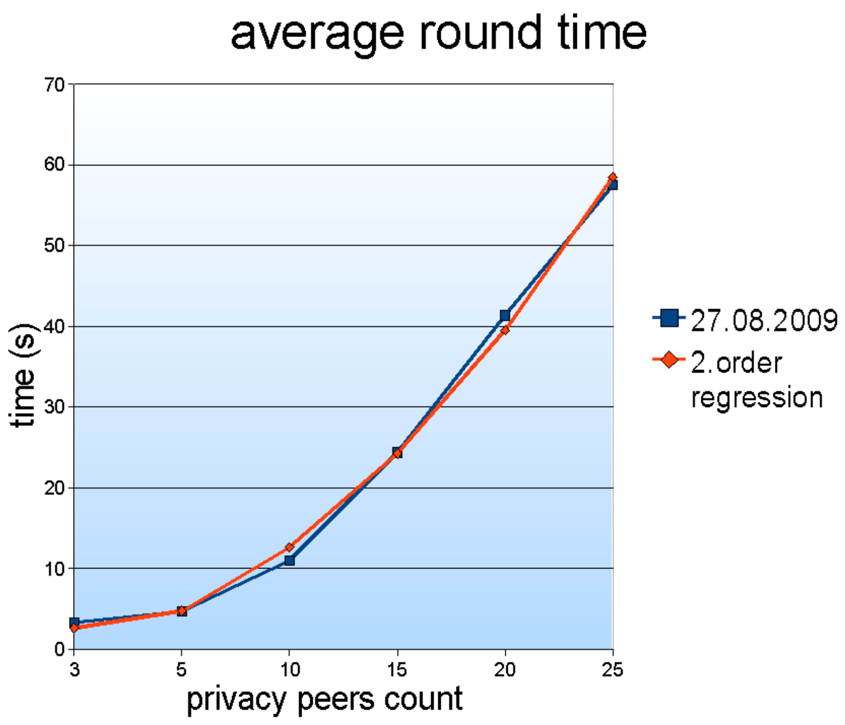

Figure 5.19.: average round time depending on the privacy peers count with a regression using a 2nd order polynomial

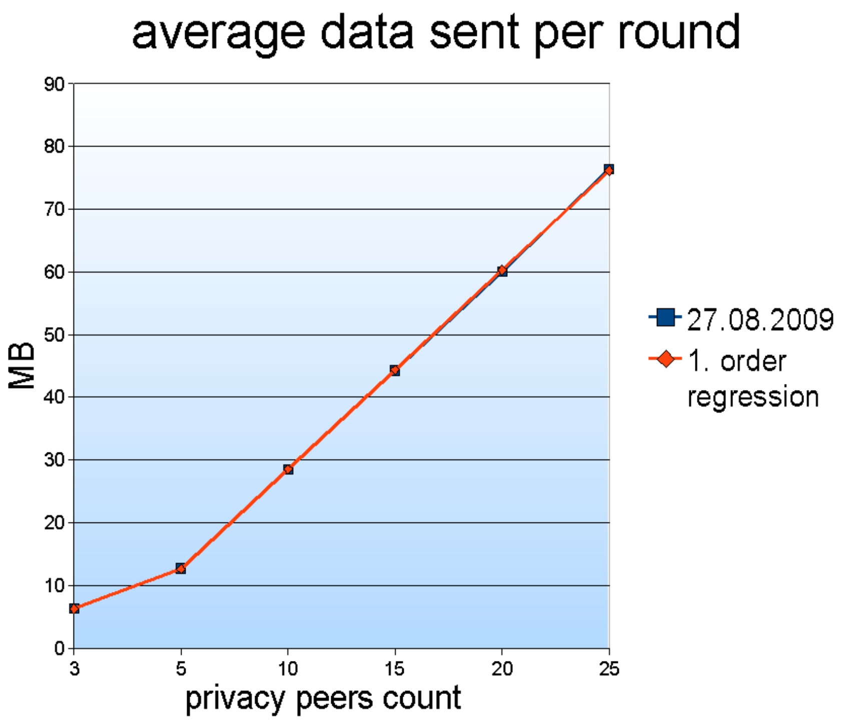

Figure 5.20.: average data sent per round by privacy peers depending on the privacy peers count with a regression using a first order polynomial

5.6. Dependence on Features Count Per Peer

5.6.1. Results

The following plots show how the user time of the privacy peers (Fig. 5.21), the average round time (Fig. 5.22), the maximum memory usage of privacy peers (Fig. 5.23) and average data sent per round by privacy peers (Fig. 5.24) depend on the features count per peer.
The different lines of the average round time (Fig. 5.22) and maximum memory usage of privacy peers (Fig. 5.23) plots represent the results from the measurements made on 10 and 18 of August.
The line in the user time of the privacy peers (Fig. 5.21) plot is based on the average of the results from the different dates. The average was used as the results were so close that one couldn't distinguish them anyway.
The line in the average data sent per round by privacy peers (Fig. 5.24) plot is based on the measurements of August 18.
The number of peers and privacy peers was set to 3 and 24 bit features were used.
The user time and average data sent per round plots show a roughly quadratic growth. The measurement results of 10 of August in the average round time plot show a quadratic growth as well. The results of August 18 show a quadratic growth as well except for the first and last measurement. These two outliers can be explained by the varaying background load of the test machines. (During the first two measurements the privacy peers didn't seem to work on any other demanding task, during the next 3 measurements matlab jobs were running on 1 privacy peer and others were used by students. During the last testrun of the series 2 privacy peers were used by students.)
The lines in the maximum memory usage plot seem to follow a sublinear growth.

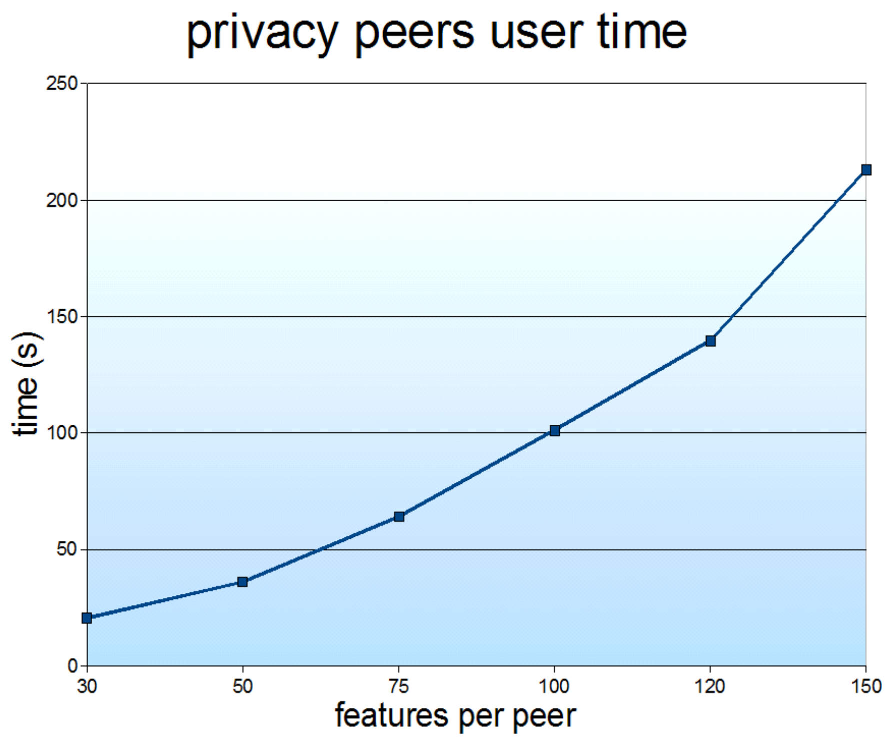

Figure 5.21.: user time of privacy peers depending on the features count per peer

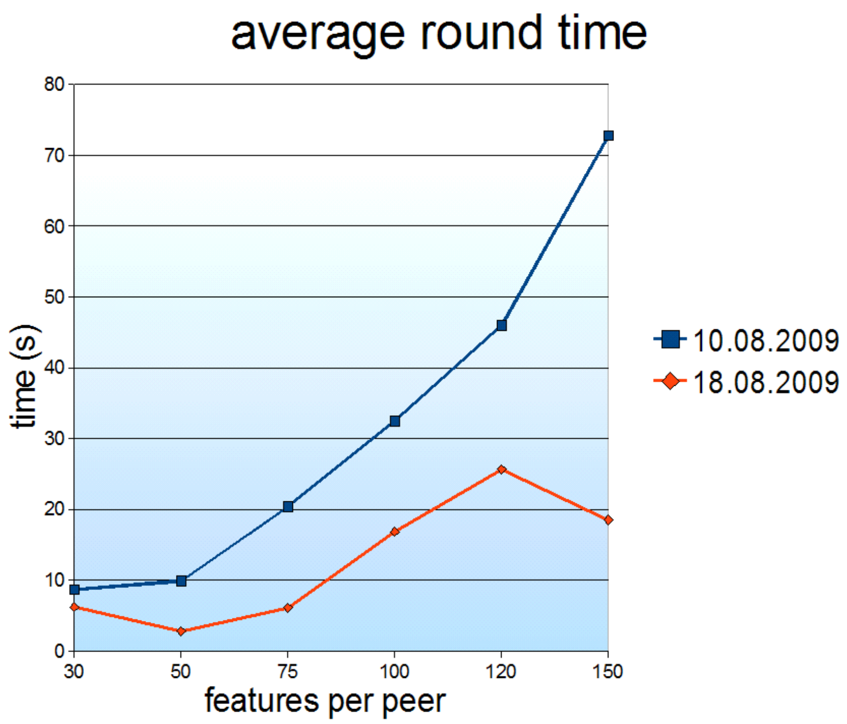

Figure 5.22.: average round time of privacy peers depending on the features count per peer

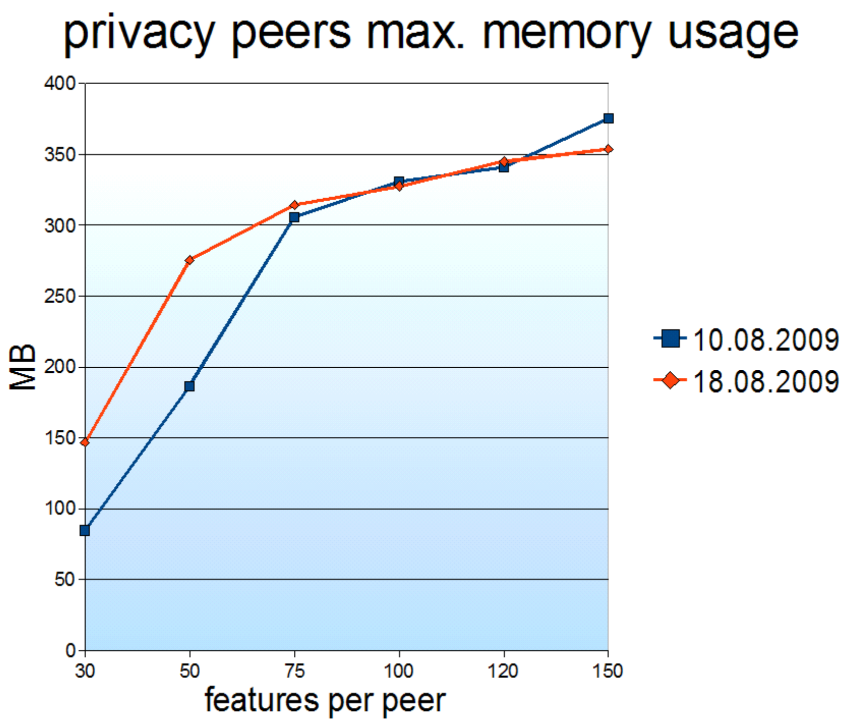

Figure 5.23.: maximum memory usage of privacy peers depending on the features count per peer

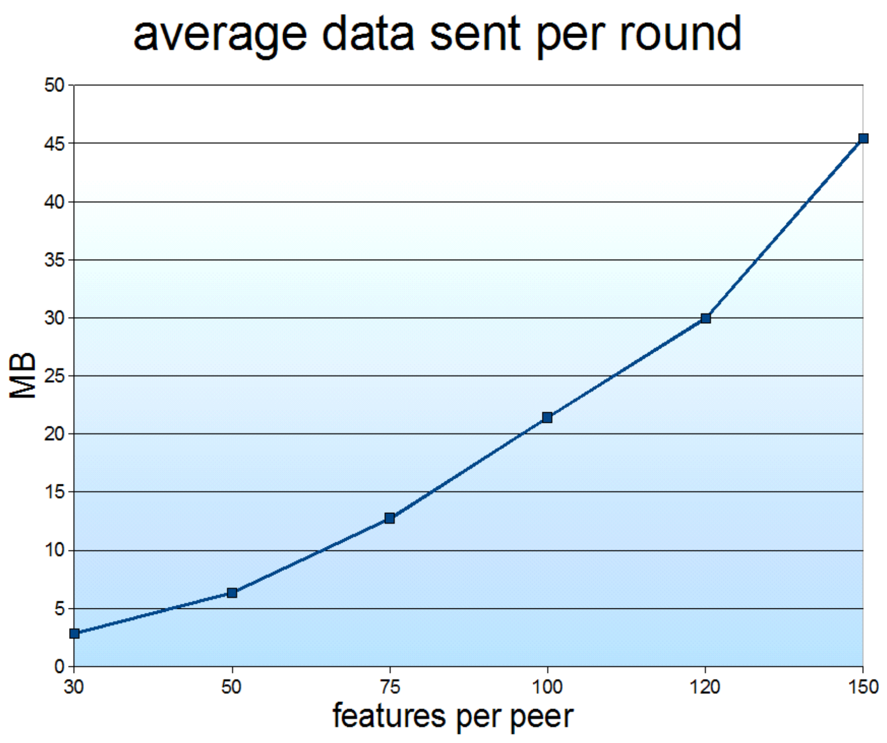

Figure 5.24.: average data sent per round by privacy peers depending on the features count per peer

5.6.2. Interpretation

We can conclude that the user time of the privacy peers, the average round time and the average data sent per round by privacy peers depend quadratically on the features count per peer. This was expected from the result of the WSIP efficiency analysis 3.6.2.
The maximum memory usage of privacy peers seems to grow sublinearly. This is rather suprising. As both measurement series show this behaviour it's unlikely that this is only due to the low quality of the results (due to the machines background load).

5.7. Dependence on Field Size

5.7.1. Results

The following plots show how the user time of the privacy peers (Fig. 5.25), the average round time (Fig. 5.26), the maximum memory usage of privacy peers (Fig. 5.27) and average data sent per round by privacy peers (Fig. 5.28) depend on the features count per peer.
The number of peers was 9, the number of privacy peers was set to 3 and the features count per peer was 50.

5.7.2. Interpretation

We can conclude that the user time of the privacy peers, the average round time, maximum memory usage and the average data sent per round by privacy peers depend logarithmically on the field size. This was expected from the result of the WSIP efficiency analysis 3.6.2.

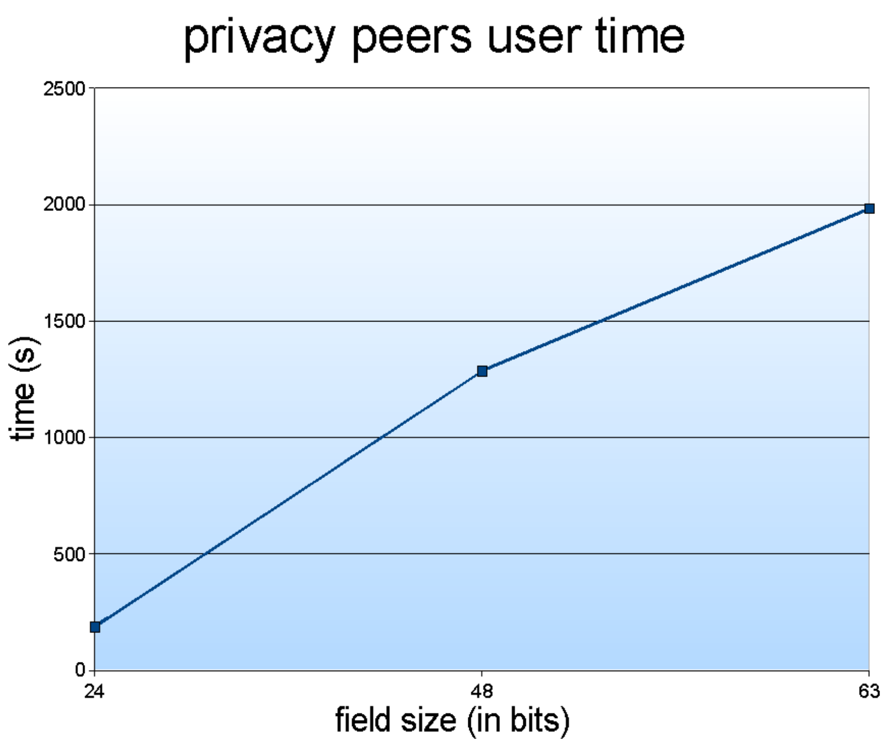

Figure 5.25.: user time of privacy peers depending on the field size

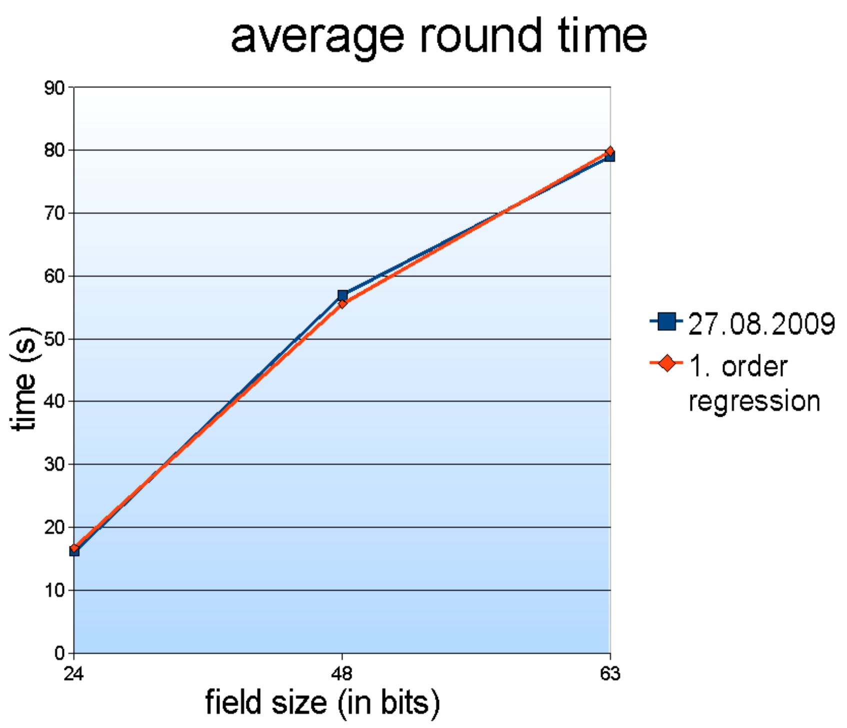

Figure 5.26.: average round time of privacy peers depending on the field size with a regression using a first order polynomial

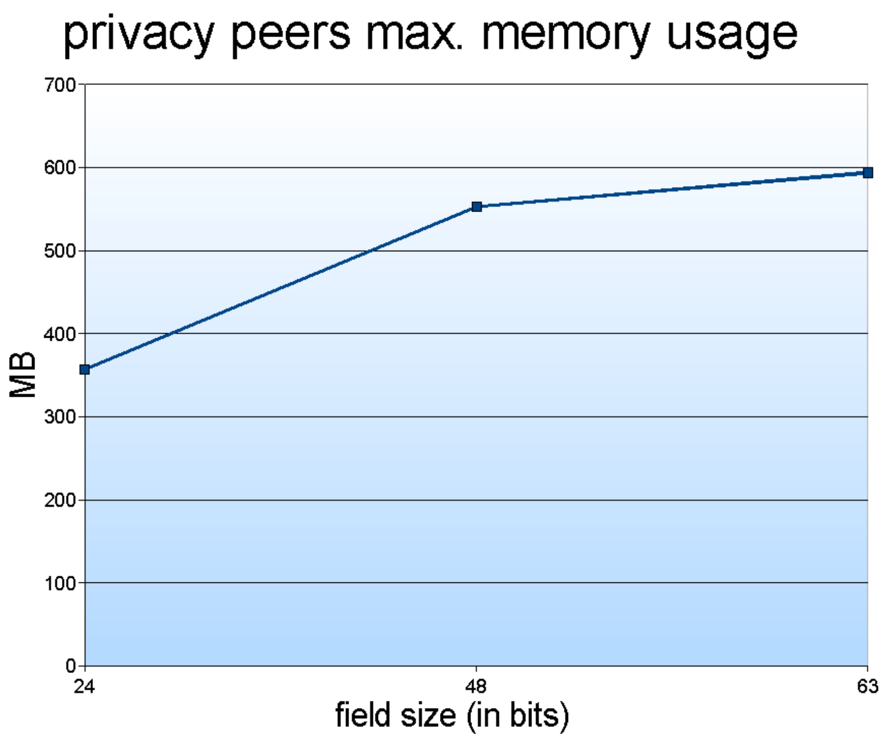

Figure 5.27.: maximum memory usage of privacy peers depending on the field size

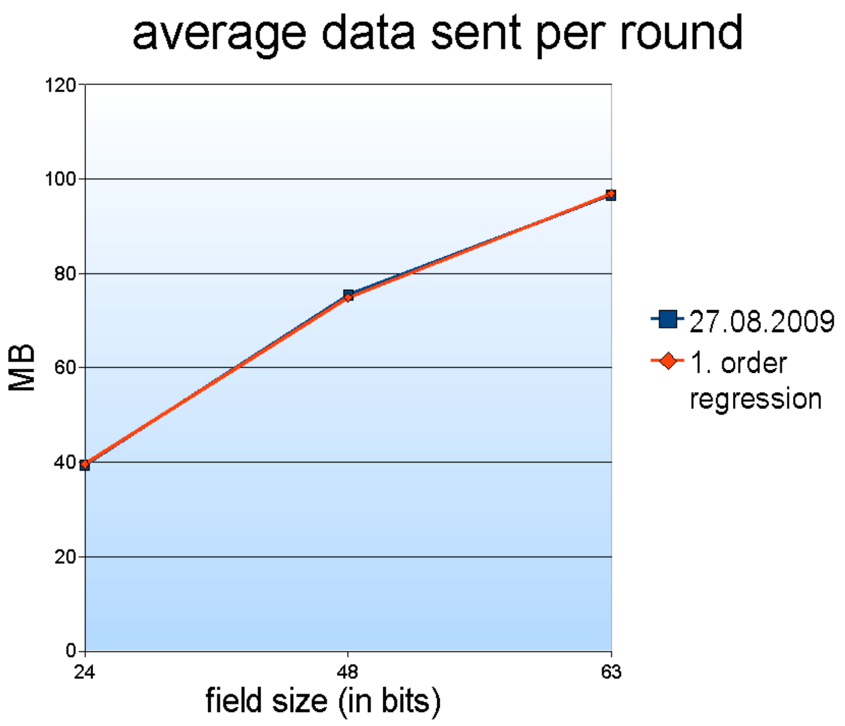

Figure 5.28.: average data sent per round by privacy peers depending on the field size with a regression using a first order polynomial

5.8. Comparison with other MPC Frameworks

The table 5.2 contains some performance results of different frameworks. VIFF is the only other framework which uses Shamir shares. ShareMind uses some special technique which restricts it to 3 privacy peers. The FairplayMP framework wasn't included in the table due to the lack of performance numbers for the operations of interest.
In general the frameworks used 5 privacy peers except for ShareMind which only supports 3. SEPIA and VIFF were measured on one of ETHs clusters while the numbers of ShareMind are taken from the research paper [11]. SEPIAs performance results were derived from a WSIP execution, therefore performance measurements with a pure benchmark protocol might be even better.

Operation	SEPIA	VIFF	ShareMind
Multiplication	95,200	326	163,000
Equal	3,800	2.3	180
LessThan	110	2.3	360
LT w/o random	500		

Table 5.2.: Performance of different MPC frameworks in number of operations per second.

The Frameworks weren't tested on the exact same machines therefore the above table only allows for an approximate comparison. There wasn't sufficient time for a detailed comparison. Nevertheless SEPIA seems to compare quite well to the other frameworks.
The LessThan operation as it is implemented in SEPIA requires bitwise shared random numbers, which are generated by the privacy peers. The most time of computing the less-than is spent generating these random numbers. But one could pre-generate them before the protocol is run, which then makes the operation much faster.
A WSIP protocol execution which takes in SEPIA 3 min. would take in VIFF (given a WSIP implementation for this framework) more than 3 days!

5.9. Conclusions

As expected the weighted set intersection protocol running time and ressource consumption is largely dominated by the many equality comparisons that have to be executed.
The running time is quadratic in the number of peers (if the min. feature count threshold increases along with the peers count), privacy peers and features count and logarithmic in the field size. The running time only depends linearly on the peers count if the difference of the min. feature count threshold to the peers count is fixed.
The average data sent by the privacy peers is quadratic in the number of peers and features count, linear in the privacy peers count and logarithmic in the field size.
The maximum memory usage of privacy peers is quadratic in the number of peers and linear in the privacy peers count. For the number of features per peer it seems to be sublinear and roughly logarithmic for the field size.
Compared to other MPC frameworks SEPIA does very well. Compared to VIFF (which uses Shamir shares as well) it is several orders of magnitude faster. Compared to ShareMind (which uses a special optimized technique which at the same time restricts it to 3 privacy peers) it is not much slower.

6. Conclusions

The literature study revealed a usefull new protocol for SEPIA. In the second part of the master thesis the protocol was specified in detail and many ideas were developed to construct the final weighted set intersection MPC protocol. In the third part the protocol was actually implemented. While doing that a package of very helpfull classes emerged (the protocol primitives) which allowed to access and use basic operations in a simple and intuitive way. In the fourth part the implemented protocol was evaluated for performance. The protocol did very well. Only the memory consumption was higher than expected. But for that I already have a potential solution as can be read in the following chapter.

7. Future Work

While obviously the protocol primitives can be extended with many other operations, there are some other things that can be addressed:

- The old protocols can be changed to use the new protocol primitives.
- The underlying code of SEPIA could be simplified and some bugs fixed.

Below are some tasks that I describe in more detail as I intended to do them.

7.1. Reducing Memory Consumption of Protocol Primitives

When doing several hundreds of thousands of operations in parallel the memory consumption might be very high. The protocol primitives could be changed, s.t. only a specified number of operations are created and executed in parallel (as far as memory consumption allows and as far it benefits in improving the network usage) while the other operations are queued.
More specifically some operations are not created immediately but rather their type and input data is stored in memory. The operations are only created later on when the previous operations are completed. For the completed operations the results can then just be stored while the rest of the operations are deleted immediately to free the memory.

7.2. Reducing Sent Data of Protocol Primitives

Currently the protocol primitives always send the shares as long values. If the largest element of the field fits into an integer the protocol primitives should send the shares as integer values instead of longs. This will greatly reduce the amount of data sent and therefore speed up all operations.

7.3. Generation of Bitwise Shared Random Numbers

The generation of the bitwise shared random numbers for some of the operations took on average 25% of the running time of WSIP, although the operations using these numbers are a minority! The BatchGenerateBitwiseRandomNumbers operation was tuned using the UnitTests but it probably should have been tuned more.
Instead of generating the bitwise shared random numbers the way it's currently implemented, one could go a totally different way. The privacy peers could probably just generate a random seed together and then generate the shared random numbers without any communication or at least greatly reduced communication.

7.4. bitwiseLessThan Operation

The bitwiseLessThan operation ($a <_B b$) can be improved. Currently if a is public log(fieldSize) multiplications are used (additionally to the use of the prefix-or subprotocol) and none if b is public. The operation can be changed that in case of a being public no furter multiplications are used just like when b is public, using $a <_B b = 1 - (b <_B (a + 1))$. Then in both cases only the prefix-or has to be computed.

Bibliography

[1] About nagios. http://www.nagios.org/about/.

[2] Free real-time netflow traffic analyzer. http://computerperformance.co.uk/HealthCheck/real_time_netflow_analyzer.htm.

[3] Going beyond the flow: Giving network engineers the tools to think, act globally.

[4] Nagios. http://nagios.sourceforge.net/images/screens/new/nagios-pnp.png.

[5] NetUP UTM. http://freshmeat.net/projects/billingsystemnetuputm/.

[6] Network flow analysis. http://iso.csusb.edu/tools/network-flow.

[7] Security manager plus. http://www.manageengine.com/products/security-manager/index.html.

[8] T. Baba and S. Matsuda. Tracing network attacks to their sources. *IEEE Internet Computing*, 6(2):20–26, 2002.

[9] M. Barisic. Sharing of traffic measurements with privacy based on multi-party computation. Master's thesis, 09 2008. http://www.tik.ethz.ch/db/public/tik/?db=publications\&form=report_single_publication\&publication_id=3189.

[10] A. Belenky and N. Ansari. On IP traceback. *IEEE Communications Magazine*, 41(7):142–153, 2003.

[11] D. Bogdanov, S. Laur, and J. Willemson. Sharemind: a framework for fast privacy-preserving computations. Cryptology ePrint Archive, Report 2008/289, 2008. http://eprint.iacr.org/.

[12] M. Burkhart, D. Brauckhoff, and M. May. On the Utility of Anonymized Flow Traces for Anomaly Detection. In *19th ITC Specialist Seminar on Network Usage and Traffic (ITC SS 19)*, October 2008.

[13] M. Burkhart, M. Strasser, and X. Dimitropoulos. Sepia: Security through private information aggregation. Technical Report 298, Computer Engineering and Networks Laboratory, ETH Zurich, Feb. 2009.

[14] M. Chapple. How helpful is the centralized logging of network flow data? 11 2008. http://searchsecurity.techtarget.com/expert/KnowledgebaseAnswer/0,289625,sid14_gci1301770,00.html.

[15] D. Ellis, J. Aiken, K. Attwood, and S. Tenaglia. A behavioral approach to worm detection. In *Proceedings of the 2004 ACM workshop on Rapid malcode*, pages 43–53. ACM New York, NY, USA, 2004.

[16] S. Frei. Network security (hs 2008) lecture slides "intrusion detection & prevention". 11 2008. http://www.csg.ethz.ch/education/lectures/network_security/hs08.

[17] S. Frei, T. Duebendorfer, G. Ollmann, and M. May. Understanding the Web browser threat. Technical Report 288, TIK, ETH Zurich, June 2008. Presented at DefCon 16, Aug 2008, Las Vegas, USA. http://www.techzoom.net/insecurity-iceberg.

[18] S. From and T. Jakobsen. Secure Multi-Party Computation on Integers. Master's thesis, Aarhus Universitet, Datalogisk Institut, 04 2006.

[19] Z. Gao and N. Ansari. Tracing cyber attacks from the practical perspective. *IEEE Communications Magazine*, 43(5):123–131, 2005.

[20] R. Gennaro, M. Rabin, and T. Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In *7th annual ACM symposium on Principles of distributed computing (PODC)*, 1998.

[21] iReasoning Networks. Sysuptime network monitor. http://www.ireasoning.com/network_monitor.shtml.

[22] S. Katti, B. Krishnamurthy, and D. Katabi. Collaborating against common enemies. In *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, pages 34–34. USENIX Association Berkeley, CA, USA, 2005.

[23] A. Kind, X. Dimitropoulos, S. Denazis, and B. Claise. Advanced network monitoring brings life to the awareness plane. *IEEE Communications Magazine*, 46(10):140–146, 2008.

[24] L. Kissner and D. Song. Privacy-Preserving Set Operations. *LECTURE NOTES IN COMPUTER SCIENCE*, 3621:241, 2005.

[25] A. Lakhina, M. Crovella, and C. Diot. Mining anomalies using traffic feature distributions. In *SIGCOMM '05: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 217–228, New York, NY, USA, 2005. ACM.

[26] R. Li and C. Wu. An unconditionally secure protocol for multi-party set intersection. *Lecture Notes in Computer Science*, 4521:226, 2007.

[27] P. Lincoln, P. Porras, and V. Shmatikov. Privacy-preserving sharing and correlation of security alerts. In *13th USENIX Security Symposium*, 2004.

[28] M. Lisa Barisic. *SEPIA Documentation*. ETH Zurich, 02 2009.

[29] Microsoft. How to use network monitor to capture network traffic. 10 2006. http://support.microsoft.com/?scid=kb%3Ben-us%3B812953\&x=18\&y=15.

[30] T. Miller. Intrusion detection level analysis of nmap and queso. 08 2000. http://www.securityfocus.com/infocus/1225.

[31] T. Nishide and K. Ohta. Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. *Lecture Notes in Computer Science*, 4450:343, 2007.

[32] J. Parekh, K. Wang, and S. Stolfo. Privacy-preserving payload-based correlation for accurate malicious traffic detection. In *Proceedings of the 2006 SIGCOMM workshop on Large-scale attack defense*, pages 99–106. ACM New York, NY, USA, 2006.

[33] M. Roughan and Y. Zhang. Privacy-preserving performance measurements. In *MineNet '06: Proceedings of the 2006 SIGCOMM workshop on Mining network data*, pages 329–334, New York, NY, USA, 2006. ACM.

[34] M. Roughan and Y. Zhang. Secure distributed data-mining and its application to large-scale network measurements. *ACM SIGCOMM Computer Communication Review*, 36(1):7–14, 2006.

[35] S. Schechter, J. Jung, and A. Berger. Fast detection of scanning worm infections. *Lecture notes in computer science*, pages 59–81, 2004.

[36] Securify. Use case: Leveraging flow data (netflow, jflow) for identity-based monitoring. http://www.securify.com/platform/uc_flowbased_data.html.

[37] A. Shamir. How to share a secret. *Commun. ACM*, 22:612–613, 1979.

[38] D. Xu and P. Ning. Privacy-preserving alert correlation: A concept hierarchy based approach. In *Computer Security Applications Conference, 21st Annual*, page 10, 2005.

[39] Q. Ye, H. Wang, and J. Pieprzyk. Distributed private matching and set operations. *Lecture Notes in Computer Science*, 4991:347, 2008.

[40] Q. Ye, H. Wang, and C. Tartary. Privacy-preserving distributed set intersection. In *Availability, Reliability and Security, 2008. ARES 08. Third International Conference on*, pages 1332–1339, 2008.

A. Protocol Design Ideas

A.1. A Simple Protocol

The peers and privacy peers agree on the thresholds t and $\tilde{t}$.
Each peer j selects s features ($f_i^j, 1 \leq i \leq s$) and the corresponding weights that he wants to use in the protocol. He then creates for each f_i^j the shamir shares $s[f_i^j]_1, s[f_i^j]_2, s[f_i^j]_n$ and $s[w_i^j]_1, s[w_i^j]_2, ...s[w_i^j]_n$ and sends them to the privacy peers.
For each feature f_k^l the privacy peers create a sharing of a random value $\bar{r}_{k,l} \neq 0$ and compute $([f_i^j] - [f_k^l]) * [\bar{r}_{k,l}]$. This randomized difference is 0 if (and only if) the features are identical. This is computed for all possible feature pairs.
Then for every feature f_k^l we intialize $[\omega_k^l] = 0$, $\phi_k^l = 0$ and compute for every $i \neq k, j \neq l$:
if $([f_i^j] - [f_k^l]) * [\bar{r}_{k,l}] = 0$ then $[\omega_k^l] = [\omega_k^l] + [w_i^j]$
if $([f_i^j] - [f_k^l]) * [\bar{r}_{k,l}] = 0$ then $\phi_k^l = \phi_k^l + 1$
Then the privacy peers compute $\hat{\omega}_k^l = \big([\omega_k^l] - t\big) * [r_{k,l}]$ with a new $r_{k,l} > 0$ (and reconstruct $\hat{\omega}_k^l$). If $\hat{\omega}_k^l \geq 0$ and $\phi_k^l \geq \tilde{t}$ they reconstruct f_k^l and ω_k^l.
After the above steps were done for all f_k^l the privacy peers send the results to the peers.

Creating a Shared Random Value: Each privacy peer j selects a random r_j and creates and distibutes the Shamir shares $s[r_j]_k$. From these shares the privacy peers can then compute locally their share of $r = \sum_{j=1}^{n} r_j$ by computing $s[r]_j = \sum_{k=1}^{n} s[r_k]_j$.

The above protocol has one serious security flaw: Given a privacy peer and a peer j collaborate. If $([f_i^j] - [f_k^l]) * [\bar{r}_{k,l}] = 0$ and the privacy peer reports that to the peer j, then j knows the value of f_k^l (as it is equal to j's f_i^j) which was supposed to remain secret.
Therefore either the privacy peers must not ever collaborate with any of the peers or we have to "mix up" (see below) the feature sets (indices) s.t. the privacy peers can't say which of the peers feature is involved in any of the comparisons.
But even if we mix up the feature sets, the privacy peers will still now that x peers have some feature f_k^l. Allthough they might not know what the feature f_k^l is, this information might still be sensitive. Therefore we have to find a protocol which doesn't require to reveal if any two features are equal or not.

A.1.1. "Mixing Up" the Feature Sets

Let P be a random set of m/2 privacy peers. All privacy peers together create from their shares of all the f_i^j and w_i^j new shares for the privacy peers in P and send the new shares to them. All privacy peers together determine a new random set P' containing m/2 of the privacy peers.
The privacy peers in P agree on a (random) permutation $\pi : [1, s] \rightarrow [1, s]$, create new shares of all the $f_{\pi(i)}^j$ and $w_{\pi(i)}^j$ for the privacy peers in P' and send the new shares to them.
P is set to P', all privacy peers together determine a new set P' containing m/2 of the privacy peers and the above steps are repeated until no $q < m/2$ privacy peers were involved in all

the "mixing rounds". (Then no q privacy peers can reconstruct which of the original shares belong to which new shares as they missed at least one of the permutations.)
In the last resharing the shares are again constructed for all privacy peers and distributed to all of them.
Notice: As P contains m/2 privacy peers and at most $q < m/2$ privacy peers collaborate they can never reconstruct any of the shares.

It seems that finding such sets P is not that easy. By hand such sets can be found for m=5 (or less). For larger m's it might get too cumbersome. I made a brute-force implementation (in Java) to find such sets. The number of sets computed (s.t no q privacy peers can be selected s.t. at least one of them is in each set) increases rapidly and the computation time increases even more. For m<8 the number of sets is at most 35 and the computation time at most 16ms. But for m=9 (or m=10) the number of sets is 126 (122) and the computation time 516ms (640ms). For m=11 the number of sets jumps to 462 and the computation time is 157641ms (> 2.5min.)!

If the number of malicious privacy peers $q < \sqrt{m} - 1$ we can easily just split up the m privacy peers into q+1 sets P of q+1 privacy peers.
Due to the security concerns no further effort was made to find any better ways of "mixing up" the feature sets.

After the weighted set intersection protocol design was assumed to be completed a potential security flaw emerged in the "final" protocol which was solved by mixing the feature sets. The simple solution is described in section A.2.5. It can be applied here as well.

A.2. Protocol using Fermats Little Theorem

The idea behind this protocol is to have a function taking 2 arguments which evaluates to 1 if they are equal and to 0 otherwise:

$$f(x,y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{if } x \neq y \end{cases} \tag{A.1}$$

(Any function evaluating to two different but fixed values for in-/equality can be easily transformed to the above form.)
To archieve that Fermat's little theorem was used.[1] It states that in a field GF(p)[2]:

$$c^{p-1} = \begin{cases} 0 & \text{if } c = 0 \\ 1 & \text{if } c \neq 0 \end{cases} \tag{A.2}$$

With this we can construct an equal operation as follows:

$$[equal([x],[y])] = 1 - ([x] - [y])^{p-1} \tag{A.3}$$

For bad choices of p this operation might take up to $2 * \lceil log(p-1) \rceil$ multiplications. But I could find many good primes p with a low number of "1"-Bits. With these primes the operation takes between $\lceil log(p-1) \rceil + 2$ and $\lceil log(p-1) \rceil + 4$ multiplications (and $log(p)$ rounds).

[1]Inspired by the solutions to exercise 11.3a) of the course "Kryptographische Protokolle" (visited in the spring term 2008)
[2]Galois Field of prime order p

In [31] the authors present a deterministic equal operation which uses $81 * log(p)$ multiplications and takes 8 rounds. While the number of rounds is clearly better than of the operation using Fermat's little theorem, the number of multiplications is much higher.

A.2.1. The Protocol

1. All peers and privacy peers have to agree on a prime $p > |F|$ and on the thresholds t, $\tilde{t}$ and max_w.
2. Each peer j selects s features ($f_i^j, 1 \leq i \leq s$) and the corresponding weights that he wants to use in the protocol. He then creates for each f_i^j the shamir shares $s[f_i^j]_1, s[f_i^j]_2, ..., s[f_i^j]_m$ and $s[w_i^j]_1, s[w_i^j]_2, ..., s[w_i^j]_m$ and sends them to the privacy peers.
3. Then the privacy peers check each set of features contributed by a peer for duplicates. (see below)
4. Additionally the privacy peers can check that the weights contributed by the peers don't exceed a given threshold max_w. (see below)
5. The privacy peers then compute
$$\forall i, k = [1, s], i \neq k : \forall j, l = [1, n], j \neq l : [equal([f_i^j], [f_k^l])], \tag{A.4}$$
where obviously $equal([f_i^j], [f_k^l]) = equal([f_k^l], [f_i^j])$ (and only one has to be computed).
6. For each feature f_k^l the privacy peers compute
$$[\omega_k^l] = \sum_{j=1}^{n} \sum_{i=1}^{s} [equal([f_i^j], [f_k^l])] * [w_i^j] \tag{A.5}$$
$$[\phi_k^l] = \sum_{j=1}^{n} \sum_{i=1}^{s} [equal([f_i^j], [f_k^l])]. \tag{A.6}$$
7. For each feature f_k^l the privacy peers compute
$$[c_{\omega_k^l}] = t \leq [\omega_k^l] \tag{A.7}$$
$$[c_{\phi_k^l}] = \tilde{t} \leq [\phi_k^l] \tag{A.8}$$
and check if
$$[c_{\omega_k^l}] \wedge [c_{\phi_k^l}] = 1. \tag{A.9}$$
8. If the check (A.9) succeeds for a feature f_k^l the privacy peers reconstruct it.

The above protocol would take O($(n * s)^2 * log(p)$) multiplications and O($log(p)$) rounds.

Remarks

- The exponentiation with p-1 (A.4) has to be done in GF(p).
- The number of features contributed per peer doesn't have to be a fixed number s. Every peer could actually contribute a variable amount of features.

A.2.2. Threshold Checks

As the equality checks (A.4) are done in GF(p) all other calculations involving the result of the check have to be done in GF(p) too.[3] Therefore there are no negative numbers. At least we can interpret the lower half of the field as positive numbers and the upper half as negative numbers. The threshold checks could be computed as

$$\hat{\omega}_k^l = \left([\omega_k^l] - t\right) * [r_{k,l}] \in \left[0, \frac{p-1}{2} - 1\right] \tag{A.10}$$

$$\hat{\phi}_k^l = \left([\phi_k^l] - \tilde{t}\right) * [r_{k,l}] \in \left[0, \frac{p-1}{2} - 1\right], \tag{A.11}$$

with $p > max(|F|, m * max_r * 2, max_\omega * max_r * 2)$, $\omega_k^l \in [1, max_\omega]$, $r_{k,l} \in [1, max_r]$. Because $r_{k,l}$ has to be chosen s.t. there is no overflow and the "sign" is preserved the resulting randomized values might not hide the original difference (to the thresholds) as well as they should.

Ideally we wouldn't have to reveal the values itself but just the result of a sign operation like:

$$sign(x) = \begin{cases} a & \text{if } x \in \left[1, \frac{p-1}{2} - 1\right] \\ b & \text{if } x = 0 \\ c & \text{if } x \in \left[\frac{p-1}{2}, p-1\right] \end{cases} \tag{A.12}$$

If we had such an operation we could use it for the equality comparison function as well. (Something like $sign(([f_i^j] - [f_k^l])([f_k^l] - [f_i^j]))$ depending on the values a,b,c of the sign operation.)
So far I haven't found any such operation (or combination of operations) in GF(p). The only noticeable thing was that $x * 2$ resulted for the lower half (where $x * 2 < p$) in a series of even numbers (0,2,4,...) and for the upper half (where $x * 2 > p$) in a series of odd numbers (1,3,5,...). But I couldn't find another operation in GF(p) that would then map the even values to one fixed value and all the odd values to another fixed value. In GF(2^b) one could just compute $x * (2^{b-1})$ s.t. even x's would result in 0 and odd ones in 2^{b-1}. But then again in GF(2^b) I couldn't find any combination of operations that would map the lower half to even values and the upper half to odd ones (or vice versa).

Threshold Check Using Polynomials: An alternative way to check for "small" thresholds $\tilde{t}$ is to construct a polynomial:

$$y(x) = x * (x-1) * (x-2) * \ldots * (x - \tilde{t} + 2) * (x - \tilde{t} + 1) \tag{A.13}$$

The polynomial is then evaluated and the privacy peers check if $y(\phi_k^l) = 0$. If yes the feature ϕ_k^l is within the interval $[0, \tilde{t} - 1]$ and therefore not at least $\tilde{t}$.
The field used for the Shamir Sharing should be of prime order. Otherwise the above product might accidentally be a multiple of the group order. Furthermore after evaluating the polynomial instead of directly reconstructing the result it should be compared to 0 for equality. Otherwise the reconstructed result might leak information about the secret x. This has the side effect that then the result is either 0 or 1 and can therefore be used for other computations without reconstructing it before.

[3]Because all the operations on the Shamir Shares always have to be in the same finite field, that was used to create the shares too. To create new shares we have to use the old shares and therefore the operations to create the new shares have to be in GF(p) too and we can't create new shares in e.g. GF(2^b) either.

This check requires $O(\tilde{t} + log(p))$ multiplications and $O((n * s)^2 * (\tilde{t} + log(p)))$ in total for all features. At least the multiplications can be done in $O(log(\tilde{t}) + log(p))$ rounds as follows: first the multiplications $x * (x-1)$, $(x-2) * (x-3)$, ... are done; then the multiplications $(x * (x-1)) * ((x-2) * (x-3))$, $((x-4) * (x-5)) * ((x-6) * (x-7))$, ... are done; etc. (At last the exponentiation is done.)

Threshold Checks Using Bit Information: There is an article [31] which solves the problem of doing the threshold checks (A.7) and (A.8) without revealing anything about the value. The main idea of the protocols is that the "bit-operations" can be done on the GF(p) Shamir shared value a by computing a bit-wise sharing of a random value r, computing GF(p) Shamir shares of it, computing and reconstructing $c = a + r$ and then doing some computations on r and c.
There are 3 protocols in the article that we could use for the threshold checks:

- Least Significant Bit: Notice that if $p > max(|F|, 2 * m, 2 * max_\omega)$ holds, $2 * (x - y)$ is even if $x \geq y$ and odd otherwise (read the rest of this subsection for details).
  Therefore the privacy peers check:
  $$LSB\left(2 * \left([\omega_k^l] - t\right)\right) = 0 \tag{A.14}$$
  $$LSB\left(2 * \left([\phi_k^l] - \tilde{t}\right)\right) = 0 \tag{A.15}$$
  with $p > max(|F|, 2 * m, 2 * max_\omega)$. The checks can be done (in parallel) using the LSB protocol. The outputs are a bit each. The privacy peers can then compute the OR of these bits to decide if they reconstruct the feature or not. All this requires $2 * n * s * (93log(p) + 1) + n * s$ multiplications and 14 rounds.

- Interval Test: Using the interval test protocol the privacy peers can check:
  $$(t - 1) < [\omega_k^l] < (max_\omega + 1) \tag{A.16}$$
  $$(\tilde{t} - 1) < [\phi_k^l] < (n + 1) \tag{A.17}$$
  with $p > max(|F|, m, max_\omega)$. These checks can be done in parallel with the interval test protocol. The privacy peers can then compute the AND of these bits to decide if they reconstruct the feature or not. All this requires $2 * n * s * (110log(p) + 1) + n * s$ multiplications and 14 rounds.

- Comparison: Using the comparison protocol the peers can directly check:
  $$(t - 1) < [\omega_k^l] \tag{A.18}$$
  $$(\tilde{t} - 1) < [\phi_k^l] \tag{A.19}$$
  with $p > max(|F|, m, max_\omega)$. The comparisons can be done in parallel. The privacy peers can then compute the AND of the output bits. All this requires $2 * n * s * (186 * log(p) + 5) + n * s$ multiplications (without the LSB check for $[t - 1 < p/2]_p$ and $[\tilde{t} - 1 < p/2]_p$) and 16 rounds. Notice that the comparison protocol is intended to compare two shared secrets, but our bounds here are public. Therefore this protocol is probably overkill.

A.2.3. Input Verification

For the protocol to work, it's essential that the input sets of the peers have no duplicates and the weights don't exceed a certain threshold. The privacy peeers can verify that these condition holds.

Check for feature duplicates

If a peer j submits two identical features $f_k^j = f_l^j, k \neq l$ that would disturb the computation of the total feature count ϕ_k^j. Which in turn could be used to maliciously manipulate the disclosure of features and their weights.
To avoid duplicate features in the set of features delivered by (any one) peer j the privacy peers compute

$$\forall k = 1, ..., s : \forall l \neq k, l = 1, ..., s : equal([f_k^j], [f_l^j]). \tag{A.20}$$

If for any pair the above computation yields 1 the peer j is disqualified. This duplicate check requires O($s^2 * log(p)$) multiplications.

Weight Bound Check

To check that the weight of a feature doesn't exceed a threshold max_w the privacy peers compute for each feature weight w_i^j submitted by a peer

$$w_i^j \leq max_w. \tag{A.21}$$

A.2.4. Efficiency Improvements

Parallelizing Multiplications

The many multiplications used for computing $\left([f_i^j] - [f_k^l]\right)^{p-1}$ (for all indices) can be done in parallel. Each privacy peer can compute $\forall i, k, i \neq k : \forall j, l, j \neq l : [f_i^j] - [f_k^l]$ and then from that compute the $(n * s)^2$ values for the first multiplication round of computing above power, put it all in one message and send it. The $(n * s)^2$ values for the next multiplication round can again be packed into one message. This is done till all O($log(p)$) multiplication rounds are done.

Feature Space Reduction

To reduce the number of multiplications the feature space could in some special cases be reduced. E.g: Instead of using all theoretically possible ports as feature space the peers could agree to only use some relevant subset of the ports like 20,21,22,23,80,135,136,445. The peers could agree on the sorted list of ports to use. Each port of the sorted list is mapped to it's position in the list. E.g: 20 → 1, 21 → 2, ... which in this example results in a feature space of [1,8]. This could be adapted for other cases.

Equality Comparisons Reduction

Fairly late in the implementation phase I had an idea to reduce the amount of equality comparisons used. Depending on $\tilde{t}$ the number of equality comparisons can be reduced dramatically. We select $n - \tilde{t} + 1$ input sets. Then instead of comparing every feature with every other feature we just compare every feature to all the features in the selected sets. This works because if a feature has to be in at least $\tilde{t}$ input sets it automatically has to be in at least one of the $n - \tilde{t} + 1$ input sets. (Hence the total count and weight is computed correctly for at least one occurence of any feature that might occur in the weighted intersection.)
Obviously $n - \tilde{t} + 1 \leq n$ as $\tilde{t} \geq 1$. Therefore in general we have to do less equality comparisons. (In the worst case all features still have to be compared to all features from all other sets.) With this idea the number of equality comparisons is reduced from $(ns) * (ns - 1)/2$ to $(d * (d - 1)/2 + d * (n - d)) * s^2$, where $d = n - \tilde{t} + 1$.

A.2.5. Security Improvements

For some reason it might be desired to hide the features which are in the intersection from the privacy peers or hide which peers had which features in their input sets. The following two protocol modifications explain how this can be archieved.

Report index instead of reconstructing feature

Instead of reconstructing f_i^j the privacy peers could just report the index i to peer j which then knows which feature is in the intersection and has a weight above the threshold. (This idea was mentioned in section "2.4 Set Intersection Computation (BETA)" of the sepia documentation. [28]) Then the privacy peers wouldn't learn which features f_i^j are in the weighted set intersection without cooperating with the peers.
Of course we then have to do all the computations for all the f_i^j and therefore can't use the efficiency improvement mentioned above.

Mixing Feature Sets

The protocol as stated above has a potential security flaw: if the privacy peers reconstruct a feature f_i^j, they know that peer j has that feature in his input set. Furthermore the privacy peers know for all the reconstructed features which peers had them in their input sets and they know of each peer how many features of their input set were not in the weighted set intersection. (If the peers shall get to know which other peers had the reconstructed features in their input sets this isn't a problem. Instead the privacy peers should then send the peers the information about which other peers have the reconstructed features in their input sets.)
Fortunately there is a rather simple solution to that problem. The features of the different input sets have to be mixed up. For each i'th feature and for all possible pairs of input sets $j \neq k$ the privacy peers have to execute the following probabilistic swap operation:

$$f_i^{j'} = f_i^j * r + f_i^k * (1 - r) \tag{A.22}$$

$$f_i^{k'} = f_i^k * r + f_i^j * (1 - r) \tag{A.23}$$

Where $r \in \{0, 1\}$ is a random bit. If this probabilistic swap operation is executed for each possible pair of sets. It's afterwards equally likely that feature $f_i^{j'}$ is from any of the original input sets.
This probabilistic swap operation requires the creation of a random bit and 4 multiplications. Each feature requires $n * (n - 1)/2$ swaps. The total costs are about $2 * s * n^2$ multiplications plus the costs of creating $s * n^2/2$ random bits. $s * n/2$ swaps can be done in parallel (as we can pick that many pairs of features without overlap). Therefore all the swaps together require $n - 1$ rounds (given the random bits were created before; otw. $n + 1$ rounds are required).

A.3. Protocol using Polynomials for Set Encoding

In most of the related work I read [26, 40, 39] the sets of the peers were encoded as polynomials. But none of them created a weighted set intersection protocol as described here. They all just did normal set intersection. The work by Kissner and Song [24] mentions that multisets could be endcoded in polynomials too.
For our purposes the peers shall encode their sets as follows (s.t the roots of the polynomials are

the encoded features). Peer j encodes his set of features F^j in a weight and count polynomial:

$$\omega^j(x) = \prod_{f_i^j \in F^j} \left(x - f_i^j\right)^{w_i^j} = a_0^j + a_1^j * x + a_2^j * x^2 + ... + a_{s*max_w}^j * x^{s*max_w} \tag{A.24}$$

$$\phi^j(x) = \prod_{f_i^j \in F^j} \left(x - f_i^j\right) = b_0^j + b_1^j * x + b_2^j * x^2 + ... + b_s^j * x^s \tag{A.25}$$

The coefficents a_i^j and b_i^j are then encrypted with a homomorphic encryption function for which $E(a) * E(b) = E(a * b)$ holds. These encrypted coefficients $E(a_i^j)$ and $E(b_i^j)$ are then sent to the privacy peers.
The privacy peers then have to compute the (encrypted) coefficients of (the union of the polynomials)

$$\omega(x) = \prod_{j=1}^{n} \omega^j(x), \tag{A.26}$$

and

$$\phi(x) = \prod_{j=1}^{n} \phi^j(x). \tag{A.27}$$

This takes O$(s * max_w * n)$ multiplications (sending O$(s * max_w * n * m)$ shares) and O$(log(n))$ rounds.
To check which features have a weight of at least t the privacy peers compute the (t-1)-th derivative of $\omega(x)$ and evaluate the polynomial for every possible feature. If $\omega(f) = 0$ then feature f has a total weight of at least t. For each derivation step the encrypted coefficients are modified as follows:

$$E(a_{i-1}) = i * E(a_i) \tag{A.28}$$

At least this can be done locally by every privacy peer. Still each evaluation requires every peer to send O$(s*max_w*n)$ shares and O$(s*max_w*n*|F|)$ shares for all evaluations together.
To check which features are in the input set of at least $\tilde{t}$ peers the privacy peers have to compute the $\tilde{t} - 1$-th derivative $\phi(x)$ and evaluate it too for every possible feature $f \in F$. If both checks are successfull for some feature f, the feature f is then reconstructed by the privacy peers and reported to the peers.
This protocol requires O$(n * s * max_w * max(|F|, m))$ communication and O$(log(n))$ rounds. It's very costly and it leaks to the privacy peers if a feature is not reconstructed because the total weight and/or the total count is too low.

The privacy peers could instead of evaluating $\omega(x)$ and $\phi(x)$ for each possible feature reconstruct the coefficients of the polynomials. And then evaluate the polynomials locally for every feature. Then the protocol requires O$(n * s * max_w * m)$ communication but additionally leaks the exact weight and count for the features that weren't reconstructed because one of the two was too low.

The polynomial $\omega^j(x)$ might leak information about the total weight of all features in the set as the privacy peers could check if any of the encrypted coefficients are $E(0)$.

Mainly due to the communication effort required this protocol wasn't investigated any further.

A.4. Protocol using GF(2) Shamir Shares

Another idea for a protocol is based on using only GF(2) Shamir Shares, i.e. b-bit values would be shared by b GF(2) Shamir Shares. This would be very universal and allow to implement the protocol without any security compromises as all the known operations on b-bit values could be supported. But it is expected to be extremly inefficient. Because the less-than comparison in the protocol using Fermats little theorem was so expensive this idea was then investigated further despite the efficiency doubts.

A.4.1. Basic Operations

The GF(2) Shamir shares support two operations: $x + y = x \otimes y$, $x * y = x \wedge y$. With these 2 operations all others can be derived:

operation	construction
$x \otimes y$	$x + y$
$x \wedge y$	$x * y$
$\neg x$	$x + 1$
$x \vee y$	$x + ((x + 1) * y)$
$x == y$	$(x + y) + 1$
$x < y$	$(x + 1) * y$
$x \leq y$	$((y + 1) * x) + 1$

As operations involving multiplications of 2 shares are expensive these should be reduced to a minimum.
In the following the b-bit numbers are assumed to be in two's complement representation.

Addition

Addition of 2 shared b-bit numbers: Let x_i, y_i be the bits ot the 2 numbers to be added. s is the sum and c_i are the carry bits. Then $s_i(x_i, y_i, c_{i-1}) = x_i \otimes y_i \otimes c_{i-1}$ and $c_i(x_i, y_i, c_{i-1}) = (x_i \wedge y_i) \vee (y_i \wedge c_{i-1}) \vee (x_i \wedge c_{i-1}) = (y_i \wedge (x_i \vee c_{i-1})) \vee (x_i \wedge c_{i-1})$
The 2 terms $(x_i \vee c_{i-1})$ and $(x_i \wedge c_{i-1})$ can be computed in parallel in one round.
This addition requires about 4b multiplications and 3b rounds.

Multiplication

multiplication of a shared b-bit number and a shared single bit: Let x_i be the bits of the b-bit number, y_1 the single bit and r_i the bits of the result. Then $r_i(x_i, y_1) = x_i \wedge y_1$.
This multiplication only requires b multiplications (of 2 shared secret bits) and 1 round.

multiplication of 2 shared b-bit numbers: Let x_i, y_i be the bits ot the 2 numbers to be multiplied. First one can multiply x_0 with y. Then x_1 is multiplied with y and shifted to the left by 1 bit. Then both results are added together (as two 2b-bit numbers). Then x_2 is multiplied with y and shifted to the left by 2 bits. This result again is added to the previous result. This process is repeated b-1 times.
This multiplication requires $9 * b^2$ multiplications (of 2 shared secret bits) and $6 * b^2 + b$ rounds. (The number of rounds could probably be reduced by not waiting till each computation completed before starting the next one but rather starting as soon as the first bits are ready for the next computation.)

Equality Comparison

comparison of 2 shared b-bit numbers: Let x_i, y_i be the bits of the 2 numbers to be compared, r the single result bit and c_i just some intermediary result bits. Then $c_i(x_i, y_i) = \neg(x_i \otimes y_i)$ and $r = c_1 \wedge c_2 \wedge ... \wedge c_b$.
$c_1 \wedge c_2$, $c_3 \wedge c_4$, ... can be computed in parallel in one round. Then we can compute $(c_1 \wedge c_2) \wedge (c_3 \wedge c_4)$, $(c_5 \wedge c_6) \wedge (c_7 \wedge c_8)$, ... in parallel in one round. And so on.
This comparison only requires b multiplications and $\lceil log(b) \rceil$ rounds.

Sign

To check if a b-bit number is $x \geq 0$ the most significant bit x_b is returned. (No actual operations are involved!) The returned bit is 1 if the number is negative and 0 if the number is 0 or positive. (Assuming two's complement representation.)
This sign operation comes for free!

A.4.2. The Protocol

1. The peers and privacy peers agree on a value b, s.t all features and weights can be represented by b-bit two's complement signed integers. Additionally they agree on the thresholds t and $\tilde{t}$.

2. The peers share their features f_i^j and weights w_i^j using b GF(2) Shamir shares for each value (e.g: 2*s*b shares in total).

3. Using the basic operations described in section A.4.1 the privacy peers compute:

   a)
   $$\forall i, j, k, l; j \neq l, i \neq k : [equal([f_i^j], [f_k^l])] \tag{A.29}$$
   this step requires $\binom{n*s}{2} * b \tilde{=} (n * s)^2 / 2 * b$ multiplications and $\lceil log(b) \rceil$ rounds

   b)
   $$\forall i, j, k, l; j \neq l, i \neq k : [equal([f_i^j], [f_k^l])] * [w_i^j] \tag{A.30}$$
   this step requires about $(n * s)^2 * b$ multiplications and 1 round

   c)
   $$\forall k, l : [\omega_k^l] = \sum_{j=1}^{n} \sum_{i=1}^{s} [equal([f_i^j], [f_k^l])] * [w_i^j] \tag{A.31}$$
   this step requires $(n * s) * (n * s - 1) * 4b$ multiplications and $\lceil log(n * s - 1) \rceil * 3b$ rounds

   d)
   $$\forall k, l : [\phi_k^l] = \sum_{j=1}^{n} \sum_{i=1}^{s} [equal([f_i^j], [f_k^l])] \tag{A.32}$$
   this step requires $(n * s) * (n * s - 1) * 4b$ multiplications and $\lceil log(n * s - 1) \rceil * 3b$ rounds (but it can be done in parallel with the previous step requiring no extra rounds)

   e)
   $$[c_{\omega_k^l}] = sign([\omega_k^l] - t) \tag{A.33}$$
   this step requires only local computations (as the threshold t is public)

f)
$$[c_{\phi_k^l}] = sign([\phi_k^l] - \tilde{t}) \tag{A.34}$$
this step requires only local computations (as the threshold $\tilde{t}$ is public)

g) and check if:
$$[c_{\omega_k^l}] \vee [c_{\phi_k^l}] = 0. \tag{A.35}$$
this step requires about $(n * s)^2/2$ multiplications and 1 round

h) for each feature f_k^l for which the final check (A.35) succeeds the feature is reconstructed
this step requires each privacy peer to send at most all his $(n * s) * b$ feature shares to all other privacy peers, e.g: $m * (n * s) * b$ bits communication (per privacy peer) in 1 round

i) the features in the weighted set intersection are reported to the peers

Remarks

- For the bound check every privacy peer can construct $-t$ and $-\tilde{t}$ locally and then compute the subtraction (as addition of the negative values) locally (as the thresholds are public).
- A bound check to check if the inputs are within the allowed range can be added. It would require only local computations of the privacy peers and the reconstruction of the results (similar to (A.33) and (A.34)). Therefore it could be done in only 1 additional round.
- Naturally a check that there are no duplicates in the features delivered by a peer could be added too. (Requiring about $s^2/2 * b$ multiplications and $\lceil log(b) \rceil$ rounds.)
- The number of rounds required to compute (A.32) can be reduced by using the knowledge of the max. value of the inputs (e.g: 1 bit values for the first addition round, 2 bit in the second, ...). Then we require for this step about $(\lceil log(n * s - 1) \rceil)^2$ rounds.
- The number of rounds required to compute (A.31) and (A.32) can be greatly reduced by starting each addition as soon as the first bits of the previous addition results required are computed. Then these steps only require about $3b + \lceil log(n * s - 1) \rceil$ rounds each.

All the steps of the privacy peers together require about $(n * s)^2 * 10b$ multiplications (of GF(2) Shamir shares) and $\lceil log(b) \rceil + \lceil log(n * s - 1) \rceil * 3b + 3$ rounds. With the optimizations mentioned in the remarks above the protocol requires only about O($(n*s)^2*b$) multiplications and O($\lceil log(n * s) \rceil + b$) rounds.

A.5. Summary of Protocols

Below is a table comparing the different protocols presented in this chapter. One multiplication requires sending of $O(m^2)$ messages (for all privacy peers together). Notice: A GF(2)-share requires only 1 bit while a GF(p) share requires $\lceil log(p) \rceil$ bits in a message.

protocol	security	multiplications	rounds
Simple	insecure	O($m * (n * s)^2$)	O(1)
using Fermats Little Theorem with:			
simple threshold check	leaks information about weights and feature counts	O($m * (n * s)^2 * b$)	O(b)
threshold check using polynomials	**secure**	O($m * (n * s)^2 * (b + \tilde{t})$)	O($b + log(\tilde{t})$)
LSB	**secure**	O($m * (n * s)^2 * b$)	O(b)
Interval Test	**secure**	O($m * (n * s)^2 * b$)	O(b)
Comparison	**secure**	O($m * (n * s)^2 * b$)	O(b)
using Polynomials for Set Encoding	leaks why feature isn't reconstructed	O($n * s * max_w * max(\|F\|, m)$)	O($log(n)$)
using GF(2) Shamir Shares	**secure**	O($m * (n * s)^2 * b$)	O($\lceil log(n*s) \rceil + b$)

where $b = \lceil log(p) \rceil$

Final Protocol

The secure protocols are based on Fermats Little Theorem or use GF(2) shares. The ones based on Fermats Little Theorem promised to be more efficient based on some calculations[4]. Furthermore GF(2) shares wouldn't fit into SEPIA very well as it uses shares in GF(p).
The final protocol is based on Fermats Little Theorem and uses different threshold checks depending on which is the fastest for the given task (see chapter 3 for details).

[4]The number of multiplications of the GF(2) protocol would be bigger for a meager 4 peers, 6 features per peer and 16-bit features.

B. Analysis of SEPIA Source Code

To get familar with the SEPIA source code I was assigned the task to make the Zero-Knowledge proofs optional in the existing protocols. This chapter describes the structure of the source code the way it was before I started on my master thesis. I wrote this text[1] as there was no real documentation of the source code and I needed to understand many aspects of it.

B.1. General

SEPIA distinguishes between peers and privacy peers (due to the reliance on the p4p framework for some parts). Privacy peer is often abbreviated to pp, especially in function and variable names.
There are 3 protocols: Additive, Entropy, UniqueCount
A couple of classes implement Runnable. When they are executed by a thread the run() function is called.

Observer Design Pattern

The Observer design pattern is heavily used in the existing code. Due to that many classes extend (inherit from) the Observable class or implement the Observer interface. A class extending the Observable class notifies its observers with setChanged() and notifyObservers() about any changes. Then the update() functions of the observers are executed to handle the notification by the thread who has executed the notifyObservers() function.
Some Observer > Observable relations are as follows:

- Peer > mpc instance
- PrivacyPeer > mpc instance
- MpcAdditivePeer > MpcAdditivePeerProtocol
- MpcAdditivePrivacyPeer > MpcAdditivePrivacyPeerProtocol, MpcAdditivePrivacyPeertoPPProtocol
- MpcEntropyPeerSecretHolder > MpcEntropyProtocolSecretHolderPeer
- MpcEntropyPeerNonSecretHolder > MpcEntropyProtocolNonSecretHolderPeer
- MpcUniqueCountPeer > MpcUniqueCountProtocolPeer
- MpcUniqueCountPrivacyPeer > MpcUniqueCountProtocolPrivacyPeer, MpcUniqueCountProtocolPrivacyPeerToPP

[1]as a part of a more much more detailed description of the code

Protocol Classes

The classes named Mpc*Protocol* (e.g: MpcAdditivePeerProtocol, MpcUniqueCountProtocolPrivacyPeerToPP) do not (as the name might imply) contain the entire protocol. Instead most parts of the protocols are actually implemented in the Mpc*Peer classes.
The protocol classes only contain the functions to send, wait for and receive messages. The messages and the data contained in the messages are actually processed by functions implemented in the peer classes. The protocol classes just pack the data (created by functions in the peer classses) into messages, send them, wait for messages, receive and unpack the messages. After receiving and unpacking messages the observers (peer classes) are notified to handle the data. All the state (variables) are in the peer classes too.
Therefore the Mpc*Protocol* classes might give a good high-level overview of the protocol (check out the run() functions) but the main parts are implemented in the Mpc*Peer classes.

B.2. Creation of a Privacy Peer and Message Handling

The figure B.2 shows all the classes that are involved when a uniqe count privacy peer is created and the observer/observable relations of these classes.

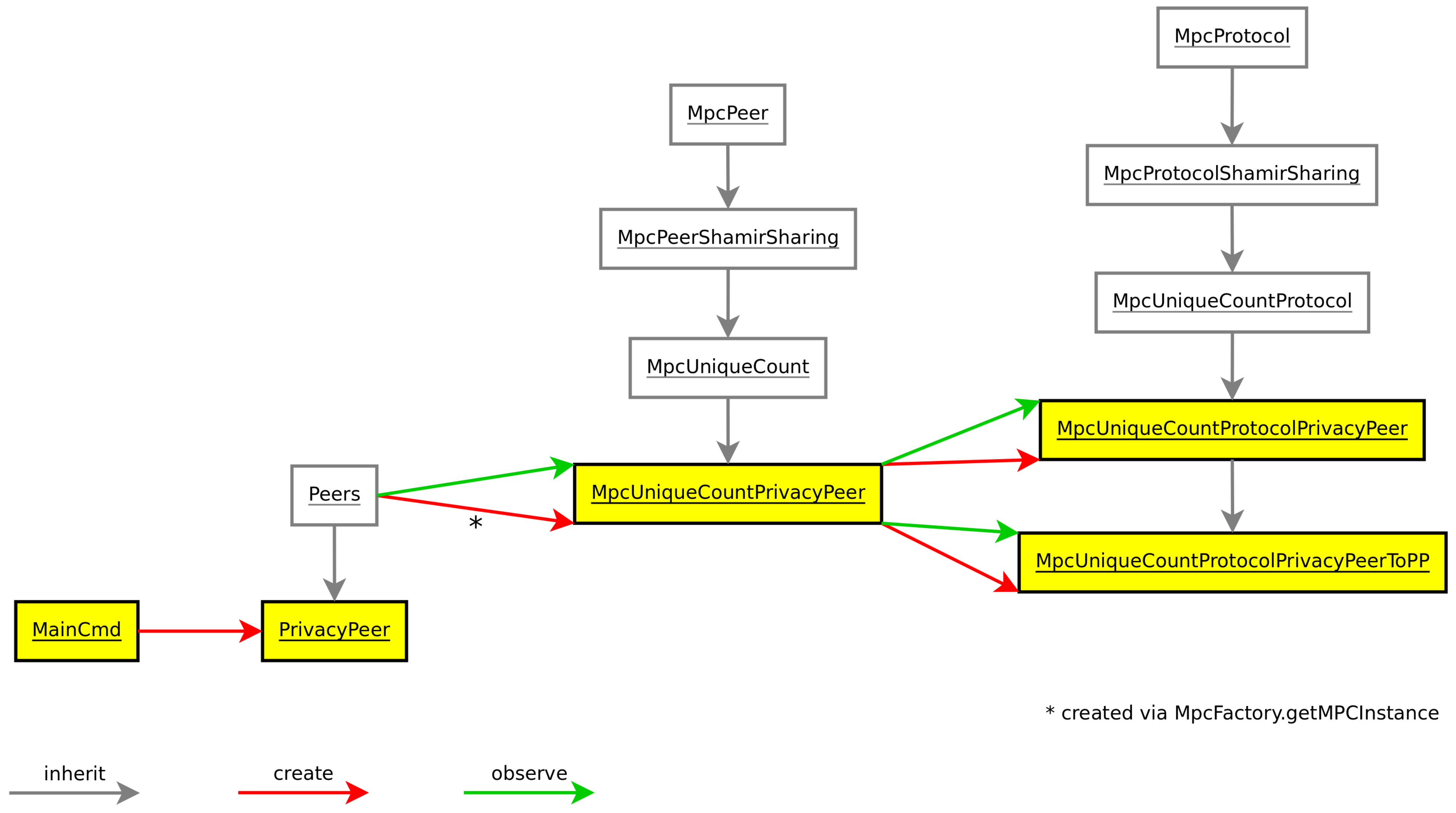

Figure B.1.: Graph depicting creation of objects for Unique Count Privacy Peer and observer/observable relations.

If the protocol threads (MpcUniqueCountProtocolPrivacyPeer and MpcUniqueCountProtocolPrivacyPeerToPP) receive messages (over the network) they don't handle them themselves but report them to their observers (e.g: a MpcUniqueCountPrivacyPeer instance).
MpcUniqueCount's update function (inherited by MpcUniqueCountPrivacyPeer) calls differnet functions from the classes MpcUniqueCountPrivacyPeer, MpcUniqueCount and MpcPeer depending on the message type. For MpcUniqueCountMessages it calls notificationReceived (of MpcUniqueCountPrivacyPeer) which either calls processPPtoPPUpdateNotification or processPPtoPeerUpdateNotification.
processPPtoPPUpdateNotification calls messageReceived. messageReceived calls either processVerificationTruncationMessage, processVerificationFinalShareMessage (+ startNewMultiplicationRound), processTruncationMessage or computeUniqueCounts (among others) depending on the received message.
As one can see, due to all the inheritance and observer/observable pattern usage following a message to grasp its effects can be daunting.

The diagram B.2 shows another example of the interaction of a peer and privacy peer of the additive protocol. The run functions of the protocol threads call the update function (via notify due to the Observer pattern) of their creators whenever they receive a messsage over the network.

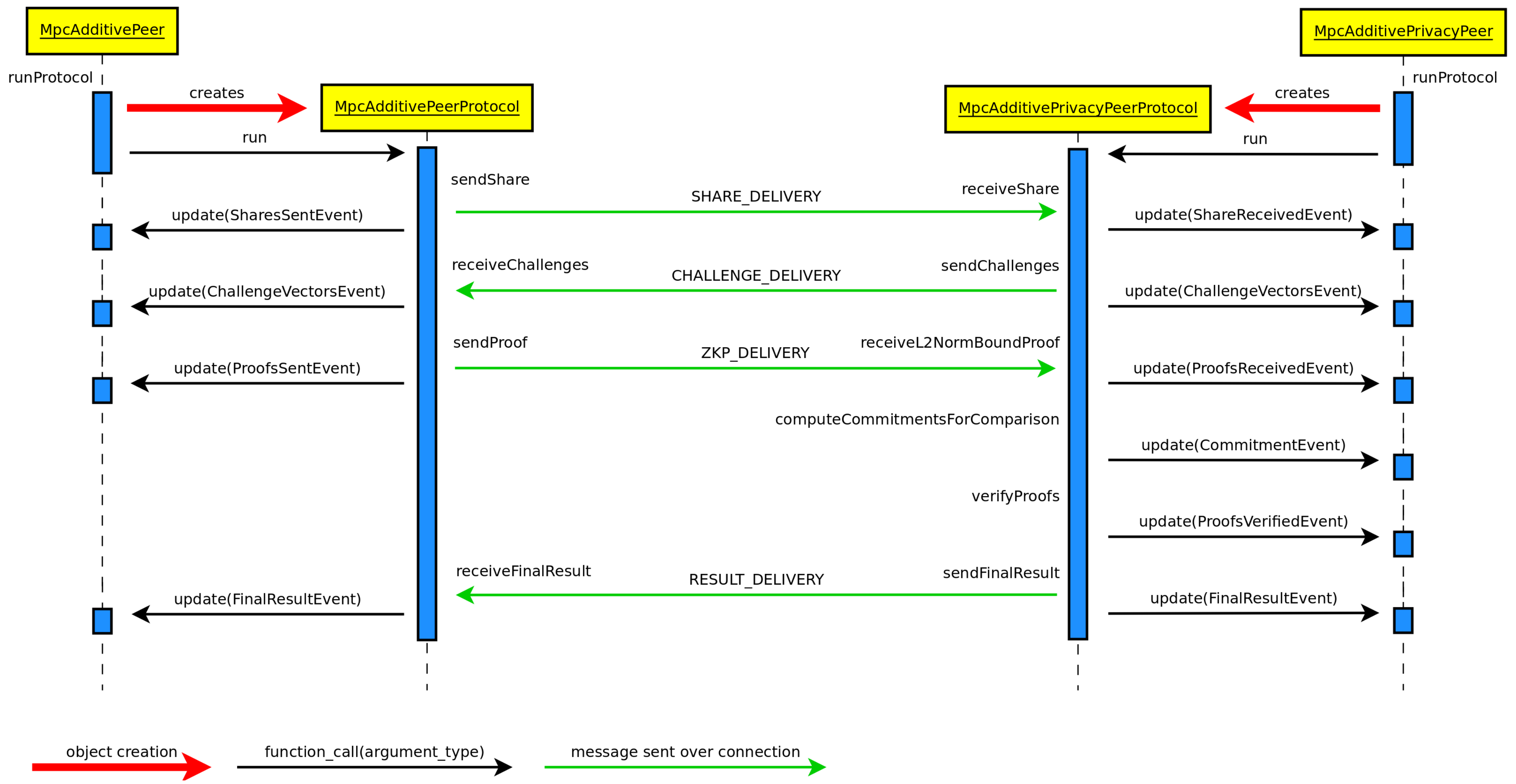

Figure B.2.: Interaction between a peer and a privacy peer.

C. Original Task Description

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für
Technische Informatik und
Kommunikationsnetze

Master Thesis
for
Dilip Many

Supervisors: Martin Burkhart, Mario Strasser

Issue Date: 02.03.2009
Submission Date: 01.09.2009

Privacy-Preserving Collaboration in Network Security

1 Introduction

A fundamental problem in network monitoring and security is the reluctance of organizations to share data related to their networks and to the traffic going through. Though data sharing is widely acknowledged to be essential for early identification of exploits, for mapping the complete vector of an attack, and most importantly for research, organizations largely refrain from sharing data. Reasons for this are: privacy violation concerns, legal risks, the competitive market, and the present lack of concrete incentives.

The state-of-the-art method for sharing traffic data is anonymization. However, traffic anonymization techniques have limitations. Firstly, the utility of anonymized traces reduces substantially, making these traces useful for a limited number of applications only [11, 7, 2]. Secondly, traffic anonymization techniques are vulnerable against fingerprinting and behavioral analysis attacks [5, 4, 9, 1, 6]; simple passive fingerprinting can deanonymize traces, which increases the risk of sharing anonymized data and the concerns people have over it. For these reasons, few organizations presently use anonymization methods to share data.

To address this problem, we are developing a tool called SEPIA [3] that uses standard techniques from secure multi-party computation (MPC) [10] to solve specific problems in network security. SEPIA supports protocols for the collaborative private computation of additive metrics, such as bytes/packets/flows per second. Further, it allows multiple parties to compute the entropy and unique count of item distributions, such as the distribution of IP addresses, ports or Autonomous Systems (AS).

2 The Task

The task of this thesis is to extend SEPIA and to deploy it in a real-world scenario, demonstrating how it improves security in a multi-domain network.

The task is split into three major subtasks: (i) literature study on Internet monitoring and network anomaly detection, (ii) extending SEPIA, (iii) Deploying SEPIA in a real-world scenario.

2.1 Literature study

The student should actively search for and study secondary literature. The goal is to identify common tasks in the domains of network monitoring or anomaly detection (e.g., the identification of common heavy-hitters) that could profit from the privacy guarantees offered by SEPIA. A short survey covering the findings should be written.

2.2 Extending the SEPIA framework

Based on the findings of the survey, a task is identified for which a privacy-preserving protocol is developed. The framework is extended as follows:

- The outlined protocol is implemented as a SEPIA module.
- Optionally, the entropy computation module is generalized to support values for parameter q that are rational and negative. This implies the extension of the secret sharing implementation to support rational numbers.

2.3 Deploying SEPIA in a realistic scenario

So far, SEPIA has only been evaluated with regard to performance. The goal of this task is to demonstrate the usefulness of SEPIA in a real-world scenario. An example for such a scenario would be a planetary-scale deployment on PlanetLab [8]. This task includes

- Definition of a realistic scenario. Data should be collected and shared in near real-time. The purpose of the application and the type of data collected should be defined.
- SEPIA must be enabled to work on continuous data feeds and not just on static input files.
- In the scenario, SEPIA is just a component. Surrounding bits and pieces that complete the scenario need to be implemented. This includes, for instance, the collection of data and transformation into input files for SEPIA or the displaying of network state and/or alarms.

3 Deliverables

The following results are expected:

- Short survey on application scenarios and metrics used in network monitoring and anomaly detection
- The design of a privacy-preserving protocol that extends the capabilities of SEPIA
- The implementation of the developed protocol
- Evaluation of the application in a real-world-scenario
- A final report, i.e. a concise description of the work conducted in this project (motivation, related work, own approach, implementation, results and outlook). The abstract of the documentation has to be written in both English and German. The original task description is to be put in the appendix of the documentation. The documentation needs to be delivered at TIK electronically. The whole documentation, as well as the source code, slides of the talk etc., needs to be archived in a printable, respectively executable version on a CDROM.

4 Assessment Criteria

The work will be assessed along the following lines:

1. Knowledge and skills
2. Methodology and approach
3. Dedication
4. Quality of Results
5. Presentations
6. Report

5 Organizational Aspects

5.1 Documentation and presentation

A documentation that states the steps conducted, lessons learned, major results and an outlook on future work and unsolved problems has to be written. The code should be documented well enough such that it can be extended by another developer within reasonable time. At the end of the project, a presentation will have to be given at TIK that states the core tasks and results of this project. If important new research results are found, a paper might be written as an extract of the project and submitted to a computer network and security conference.

5.2 Dates

This project starts on March 2, 2009 and is finished on September 1, 2009. It lasts 6 months in total. At the end of the second week the student has to provide a time schedule for the theses. It will be discussed with the supervisors.

Two intermediate presentations for Prof. Plattner and all supervisors will be scheduled 2 and 4 months into this project.

A final presentation at TIK will be scheduled close to the completion date of the project. The presentation consists of a 20 minutes talk including 5 minutes for questions. Informal meetings with the supervisors will be announced and organized on demand.

5.3 Supervisors

Martin Burkhart, burkhart@tik.ee.ethz.ch, +41 44 632 56 63, ETZ G95
Mario Strasser, strasser@tik.ee.ethz.ch, +41 44 632 74 61, ETZ G60.1

References

[1] T. Brekne, A. Årnes, and A. Øslebø. Anonymization of IP traffic data: Attacks on two prefix-preserving anonymization schemes and some proposed remedies. In *Workshop on Privacy Enhancing Technologies*, pages 179–196, 2005.

[2] M. Burkhart, D. Brauckhoff, and M. May. On the Utility of Anonymized Flow Traces for Anomaly Detection. In *19th ITC Specialist Seminar on Network Usage and Traffic (ITC SS 19)*, October 2008.

[3] M. Burkhart, M. Strasser, and X. Dimitropoulos. SEPIA: Security through Private Information Aggregation. Technical Report 298, Computer Engineering and Networks Laboratory, ETH Zurich, Feb. 2009.

[4] S. Coull, C. Wright, A. Keromytis, F. Monrose, and M. Reiter. Taming the devil: Techniques for evaluating anonymized network data. In *15th Annual Network and Distributed System Security Symposium (NDSS)*, February 2008.

[5] S. Coull, C. Wright, F. Monrose, M. Collins, and M. Reiter. Playing devil's advocate: Inferring sensitive information from anonymized network traces. In *14th Annual Network and Distributed System Security Symposium*, February 2007.

[6] D. Koukis, S. Antonatos, and K. G. Anagnostakis. On the privacy risks of publishing anonymized IP network traces. In *Communications and Multimedia Security*, volume 4237 of *Lecture Notes in Computer Science*, pages 22–32. Springer, 2006.

[7] K. Lakkaraju and A. Slagell. Evaluating the utility of anonymized network traces for intrusion detection. In *4th Annual SECURECOMM Conference*, September 2008.

[8] PlanetLab. An open platform for developing, deploying, and accessing planetary-scale services. http://www.planet-lab.org/.

[9] B. Ribeiro, W. Chen, G. Miklau, and D. Towsley. Analyzing privacy in enterprise packet trace anonymization. In *15th Annual Network and Distributed System Security Symposium (NDSS)*, February 2008.

[10] A. Yao. Protocols for secure computations. In *23rd Annual IEEE Symposium on Foundations of Computer Science*, pages 160–164, 1982.

[11] J. Zhang, N. Borisov, and W. Yurcik. Outsourcing security analysis with anonymized logs. In *Securecomm and Workshops*, 2006.

February 23, 2009